Entity framework core con scaffolding del database.

Lo scorso semestre abbiamo utilizzato Entity Framework core creando noi le classi e tramite i comandi AddMigration e UpdateDatabase, abbiamo poi delegato al sistema la creazione del database e delle tabelle.

Spesso però si effettua l'operazione contraria, ovvero creaiamo prima il database e in una seconda fare, sviluppiamo l'applicazione.

In questo casi vale la pena approfittare del comando ScaffoldDb, che ci permette di eseguire un "reverse engenieering" della banca dati, creando così automaticamente gli oggetti che rappresentano una riga della tabella e il contesto di database.

Ecco lo schema che vi verrà creato quando eseguirete lo script del database. Per creare il database, aprire il file GestioneAllievi.SQL con **sql server management studio**, collegarsi al database e lanciare l'esecuzione del comando SQL completo.

Ecco gli step necessari per completare l'esercizio:

1) Configurare la stringa di connessione all'interno del file di configurazione appsetting.json

```
"ConnectionStrings": {
  "MyDb": "Server=.\\sqlexpress;Database=EFDistanza;Trusted_Connection=true;TrustServerCertificate=true;"
}
```

2) Lanciare il comando ScaffoldDb dalla Package Manager Console: attenzione..va scritto tutto su una riga !!

```
Scaffold-DbContext "Server=.\sqlexpress;Database=EFDistanza;Trusted_Connection=true;TrustServerCertificate=true;"
"microsoft.entityframeworkcore.sqlserver" -OutputDir Db -Tables Studente, Classe  -DataAnnotations -Force
```

3) Aggiungere nel motore di inversione del controllo (IOC) le informazioni necessarie per permettere l'instanziamento della classe di contesto nel caso in cui questa venga richiesta durante l'evasione della richiesta HTTP.

```
builder.Services.AddDbContext<EfdistanzaContext>(op =>
  op.UseSqlServer(builder.Configuration.GetConnectionString("MyDb")));
```

4) Creare un controller EfDistanzaController, dove inseriremo i metodi dei punti seguenti. Il controller deve implementare un costruttore specifico che ha come parametro un oggetto dello stesso tipo della classe di contesto: questo fa in modo che in fase di costruzione dell'oggetto, il motore di inversione del controllo attiva prima un'istanza del contesto di banca dati e passa l'istanza come parametro, scaricando il programmatore dalla creazione di questa istanza.

```
public class EfDistanzaController : Controller
{
    private readonly EfdistanzaContext _ctx;

    0 references
    public EfDistanzaController(EfdistanzaContext ctx)
    {
        _ctx = ctx;
    }

    0 references
    public IActionResult Index()
```

5) Creare una procedura che ritorna tutti gli allievi (GetStudenti) ordinati per cognome, nome.

```
public string GetStudenti()
{
    var studenti = _ctx.Studentes
        .OrderBy(a=>a.Cognome).ThenBy(a=>a.Nome).ToList();
    // questo genera una lista .. la dobbiamo trasformare in una stringa ..
    var studenti1 = studenti
        .Select(a => $"Cognome:{a.Cognome}, nome: {a.Nome}," +
        $" classe:{a.ClasseId}")
        .ToList();
    // e infine la lista la trasformiamo in una stringa, dove gli elementi
    // sono separati dal carattere che permette di andare a capo.
    var ris = string.Join(Environment.NewLine, studenti1);
    return ris;
}
```

6) creare un'azione che prende un nome, cognome e classeId e registra un nuovo allievo. La procedura la dovremo chiamare con il seguente URL:
https://localhost:xxxx/EfDistanza/AggiungiStudente?Nome=Abc&Cognome=Efg&ClasseId=2
L'azione ritorna una stringa come questa: "*Creato lo studente numero 13: Efg Abc, classeId 2*
"

```
public string AggiungiStudente(string nome, string cognome, int classeId)
{
  // creaiamo un'istanza di studente
  var studente = new Studente
  {
    Cognome = cognome,
    Nome = nome,
    ClasseId = classeId
  };
  _ctx.Studentes.Add(studente);
  _ctx.SaveChanges();

  return $"Creato lo studente numero {studente.StudenteId}" +
    $": {studente.Cognome} {studente.Nome}, " +
    $"classeId {studente.ClasseId} ";
}
```

6) Creare un'azione che prendendo un numero di utente da cancellare, procede con la cancellazione dello stesso: https://localhost:xxxx/EfDistanza/CancellaStudente/23. La procedura deve ritornare una stringa come quella che segue: "Cancellato lo studente numero 23" nel caso in cui lo ha trovato e cancellato, altrimenti deve ritornare una stringa con il dettaglio dell'errore.

```
public string CancellaStudente(int? id)
{
  try
  {
    // cerchiamo lo studente con questo numero di studenteId
    var studente = _ctx.Studentes.FirstOrDefault(a => a.StudenteId == id);
    if (studente != null)
    {
      _ctx.Studentes.Remove(studente);
      _ctx.SaveChanges();
      return $"Cancellato lo studente {studente.StudenteId}";
    } else
    {
      return $"Attenzione: lo studente {id} non esiste sul database !!";
    }
  }
  catch (Exception ex)
  {
    return $"Eccezione con il messaggio {ex.Message}";
  }
}
```

7) Creare un'azione AggiornaStudente, che prendendo come parametri facoltativi la classeId, il nome e il cognome, aggiorna i dati di questo studente. Ad esempio, potremmo chiamare l'azione con i seguenti dati (https://localhost:xxxx/EfDistanza/AggiornaStudente/23?Nome=Abc&ClasseId=2 per aggiornare il nome e ClasseId dello studente con StudenteId = 23. Da notare che in questo caso, visto che non passiamo il cognome, questo dato non viene modificato.

```
public string AggiornaStudente(int? id, string cognome, string nome, int? classeId = null)
{
  try
  {
    // cerchiamo lo studente con questo numero di studenteId
    var studente = _ctx.Studentes.FirstOrDefault(a => a.StudenteId == id);
    if (studente != null)
    {
      if (cognome != null)
        studente.Cognome = cognome;
      if (nome != null)
        studente.Nome = nome;
      if (classeId != null)
        studente.ClasseId = (int)classeId;
      _ctx.SaveChanges();
      return $"Aggiornato lo studente {studente.StudenteId}";
    }
    else
    {
      return $"Attenzione: lo studente {id} non esiste sul database !!";
    }
  }
  catch (Exception ex)
  {
    return $"Eccezione con il messaggio {ex.Message}";
  }
}
```

8) Sviluppare autonomamente l'azione che permette di aggiungere una nuova classe tramite il comando https://localhost:xxxx/EfDistanza/AggiungiClasse?NomeClasse=SGT1A&AnnoInizio=2023

9) Scrivere il codice dell'azione https://localhost:xxxx/EfDistanza/CancellaClasse/23 che permette di cancellare la classe con ClasseId = 23

10) Scrivere il codice dell'azione https://localhost:xxxx/EfDistanza/CancellaClasse?NomeClasse=SGT1A&AnnoInizio=2023 che permette di cancellare la classe con questi dati. Piccolo aiuto: potete scrivere il comando di ricerca del record da cancellare utilizzando qualche cosa di simile .FirstOrDefault(a=>a.NomeClasse = nomeClasse && a.AnnoInizio==annoInizio)

11) Sviluppare l'azione Scrivere il codice dell'azione https://localhost:xxxx/EfDistanza/AggiornaClasse/23?AnnoInizio=2025 che permette di modificare l'anno di inizio della classe con Id = 23