9. Adicione uma propriedade do tipo Arma chamada Arma na classe Personagem. Necessário o using System.Text.Json.Serialization;

```
[JsonIgnore]
4 references
public Usuario Usuario { get; set; }

[JsonIgnore]
0 references
public Arma Arma { get; set; }
```

- Perceba que estamos colocando a anotação *JsonIgnore* acima da propriedade desta classe Personagem. Isso será necessário para que o EntityFramework não caia em loppings de carregamentos nos métodos Get ao carregar um Personagem, já que as classes Arma e Usuário também tem propriedade do tipo Personagem. Até o final desta aula, instalaremos o pacote que evita isso e poderemos remover esta anotação JsonIgnore

10. Abra a classe DataContext para alterar o método OnModelCreating, adicionando dados para o salvamento de uma Arma, relacionando cada arma a um Id de personagem existente.

```
modelBuilder.Entity<Arma>().HasData
(
    new Arma() { Id = 1, Nome = "Arco e Flecha", Dano = 35, PersonagemId = 1},
    new Arma() { Id = 2, Nome = "Espada", Dano = 33, PersonagemId = 2 },
    new Arma() { Id = 3, Nome = "Machado", Dano = 31, PersonagemId = 3 },
    new Arma() { Id = 4, Nome = "Punho", Dano = 30, PersonagemId = 4 },
    new Arma() { Id = 5, Nome = "Chicote", Dano = 34, PersonagemId = 5 },
    new Arma() { Id = 6, Nome = "Foice", Dano = 33, PersonagemId = 6 },
    new Arma() { Id = 7, Nome = "Cajado", Dano = 32, PersonagemId = 7 }
);
```

11. Salve as classes alteradas e crie a migração através do comando "***dotnet ef migrations add MigracaoUmParaUm***"

- Observe no arquivo de criação da migração uma nova coluna para ser criada na tabela armas, bem como a definição da foreign key.

- Já no final do arquivo de design da migração pode ser observado as anotações de HasOne e WithOne que define a relação um para um além da anotação OnDelete que dita o comportamento da remoção de um personagem removido, deletando também a arma dele, conhecido como deleção em cascata.

12. Atualize o banco de dados através do EntityFramework (ef) para que a migração realize as alterações com o comando "***dotnet ef database update***" ou gere o script conforme aprendemos para rodar manualmente através do comando "***dotnet ef migrations script MigracaoUsuario MigracaoUmParaUm -o ./script04_TabelaArmas_Atualizacao.sql***"

**Relacionamento Many to Many**

Para fazer uso do relacionamento **muitos para muitos** criaremos uma classe de Habilidades em que poderemos ter diversas delas cadastradas, sendo que um personagem poderá ter várias habilidades e uma mesma habilidade poderá constar em vários personagens. Levando em consideração os aprendizados de banco de dados, isso criará uma terceira tabela na base de dados para juntar o id do personagem e id da habilidade. Vamos aos códigos!!!

13. Crie a classe **Habilidade** dentro da pasta Models. Note que desta vez não teremos o Personagem nem a lista dele sendo declarados ainda, pois como o relacionamento é muitos para muitos, esta vinculação ficará na classe a seguir.

```
namespace RpgApi.Models
{
    0 references
    public class Habilidade
    {
        0 references
        public int Id { get; set; }
        0 references
        public string Nome { get; set; }
        0 references
        public int Dano { get; set; }
    }
}
```

14. Crie outra classe chamada **PersonagemHabilidade** na pasta Models

```
namespace RpgApi.Models
{
    0 references
    public class PersonagemHabilidade
    {
        0 references
        public int PersonagemId { get; set; }
        0 references
        public Personagem Personagem { get; set; }
        0 references
        public int HabilidadeId { get; set; }
        0 references
        public Habilidade Habilidade { get; set; }
    }
}
```