Programowanie wizualne

opracował: Wojciech Frohmberg

Tworzenie asocjacji poszczególnych krotności w Entity Framework

W Entity Framework (EF) przez fakt korzystania z relacyjnych baz danych możemy zamodelować asocjacje łączące tabele z różnymi krotnościami. Poprzez krotność asocjacji mamy na myśli z iloma wpisami będzie skojarzony pojedynczy wpis z jakiejś tabeli. I tak na przykład asocjacja [1..1] oznacza, że każdy wpis z tabeli pierwszej jest skojarzony zawsze z dokładnie jednym wpisem powiązanej asocjacją drugą tabelą, z której to każdy wpis również musi być połączony zawsze z pojedynczym wpisem z tabeli pierwszej. Dla kontrastu przykładowo asocjacja [0-1..0-N] mówi, iż każdy wpis z tabeli pierwszej może być skojarzony z co najwyżej N wpisami z tabeli drugiej, ale nie musi być skojarzony z żadnym tj. może wystąpić sytuacja, w której wpis nie ma swojego odpowiednika w tabeli drugiej, z kolei każdy wpis z drugiej tabeli może być powiązany z co najwyżej jednym wpisem z tabeli pierwszej, ale też może nie być powiązany z żadnym wpisem z tej tabeli. W ramach tego dokumentu spróbujemy zebrać informacje nt. modelowania poszczególnych rodzajów asocjacji w Entity Frameworku oraz sposobie dostępu (nawigacji) do danych w przypadku ich obrania.

CZĘŚĆ I Asocjacje one-to-one

1. Asocjacja [0-1..1] – jedna z najbardziej intuicyjnych asocjacji ponieważ reprezentuje relację posiadania (ang. „has a”). Jako, że asocjacja nie jest symetryczna (to, że obiekt A posiada obiekt B nie oznacza, a w ogólności stoi w kontrze, z posiadaniem obiektu A przez B) musimy w jej ramach wyłonić rolę „podmiotu” oraz „przedmiotu”. Podmiotem będzie ten spośród modeli, który żeby zaistniał będzie musiał określić obiekt przedmiotu na który referuje. Żeby zamodelować to na poziomie języka C# określimy dwustronne zagnieżdżenie modeli wewnątrz siebie. Np.:

```
public class Entity1
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Guid Entity2Id { get; set; }
    public Entity2 Entity2 { get; set; } = null!;
}
```

```
public class Entity2
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Entity1? Entity1 { get; set; }
}
```

Kluczowy jest tutaj fakt, że tylko jedna ze stron posiada drugą na własność tutaj Entity1 posiada Entity2 co obrazowo przedstawia fakt brak znaku zapytania przy deklaracji właściwości przechowującej tą składową. Oczywiście z racji, że zarówno typ Entity1 jak i Entity2 są typami referencyjnymi (są zdeklarowane jako klasy, a nie jako struktury) z punktu widzenia deklaracji zmiennych ich typów nie ma znaczenia czy określimy typ ze znakiem zapytania czy nie – w obu przypadkach będzie można pod daną zmienną wstawić wartość null. Dlatego potrzebujemy jeszcze dodatkowego wskazania, którym jest określenie, że w Entity1 chcemy przechować nienullowalne Id obiektu Entity2, do którego obiekt Entity1 będzie referować.

W powyższym wypadku żeby zapisać obiekt typu Entity1 w bazie danych będziemy musieli utworzyć obiekt typu Entity2 (ewentualnie znaleźć jeszcze niepowiązany z innym wpisem istniejący) i powiązać go z nim. Zamodelowaliśmy zatem nienullowalne unikalne połączenie typu klucz obcy (ang. foreign key). Połączenie w drugą stronę z kolei nie musi być reprezentowane w żaden sposób w bazie – byłoby redundantne (nadmiarowe). Dobrze

obrazuje to migracja utworzona dla powyższego kodu zaprezentowana poniżej.

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "Entity2s",
        columns: table => new
        {
            Id = table.Column<Guid>(type: "TEXT", nullable: false),
            Label = table.Column<string>(type: "TEXT", nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Entity2s", x => x.Id);
        });

    migrationBuilder.CreateTable(
        name: "Entity1s",
        columns: table => new
        {
            Id = table.Column<Guid>(type: "TEXT", nullable: false),
            Label = table.Column<string>(type: "TEXT", nullable: false),
            Entity2Id = table.Column<Guid>(type: "TEXT", nullable: false)
        },
        constraints: table =>
        {
            table.PrimaryKey("PK_Entity1s", x => x.Id);
            table.ForeignKey(
                name: "FK_Entity1s_Entity2s_Entity2Id",
                column: x => x.Entity2Id,
                principalTable: "Entity2s",
                principalColumn: "Id",
                onDelete: ReferentialAction.Cascade);
        });

    migrationBuilder.CreateIndex(
        name: "IX_Entity1s_Entity2Id",
        table: "Entity1s",
        column: "Entity2Id",
        unique: true);
}
```

Warto zwrócić uwagę, że żeby EF znalazł powiązanie pomiędzy tymi modelami wystarczy nam samo referencyjne Id:

```
public class Entity1
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Guid Entity2Id { get; set; }
}
```

```
public class Entity2
{
    public Guid Id { get; set; }
    public string Label { get; set; }


}
```

Jednak wtedy by osiągnąć efekt unikalności klucza obcego (chcemy przecież zareprezentować asocjację one-to-one a nie one-to-many) będziemy musieli w ramach kontekstu, tj. przy użyciu składni Entity Framework Fluent, określić sposób budowania modelu:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Entity1>()
        .HasIndex(e => e.Entity2Id)
        .IsUnique(true);
}
```

Jeśli chcielibyśmy utworzyć podobną asocjację bez jednoznacznego określania położenia referencyjnego Id w ramach klas modeli możemy to zrobić:

```
public class Entity1
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Entity2 Entity2 { get; set; }
}
```

```
public class Entity2
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Entity1? Entity1 { get; set; }
}
```

Jednak teraz w ramach kontekstu konfiguracja we Fluence'sie nam się nieco wydłuży:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Entity1>()
        .HasOne(e => e.Entity2)
        .WithOne(e => e.Entity1)
        .HasForeignKey<Entity1>("Entity2Id")
        .IsRequired();
    modelBuilder.Entity<Entity1>()
         .HasIndex("Entity2Id")
         .IsUnique(true);
}
```

2. Asocjacja [0-1..0-1] – symetryczna asocjacja powiązania dwóch modeli. Jeśli nie chcemy tutaj tworzyć dodatkowej tabeli (modelu) łączącej oba modele – będziemy musieli się niestety zdecydować (być może czasem sztucznie) na skorzystanie z niesymetrycznego podejścia, które zakłada przechowanie klucza w jednym z modeli. Żeby osiągnąć efekt nieobligatoryjności powiązania będziemy musieli określić, że klucz obcy jest null'owalny:

```
public class Entity1
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Guid? Entity2Id { get; set; }
    public Entity2? Entity2 { get; set; }
}
```

```
public class Entity2
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Entity1? Entity1 { get; set; }
}
```

Podobnie będziemy mogli utworzyć połączenie bez nawigacji poprzez pozbycie się właściwości nawigacyjnych:

```
public class Entity1
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Guid? Entity2Id { get; set; }
}
```

```
public class Entity2
{
    public Guid Id { get; set; }
    public string Label { get; set; }

}
```

I tu znów będziemy musieli dodać informacji o unikalności klucza obcego we Fluence'sie (ponieważ nasza asocjacja to wciąż one-to-one):

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Entity1>()
          .HasIndex(e => e.Entity2Id)
          .IsUnique(true);
}
```

Znowuż też, jeśli chcielibyśmy opóźnić określenie strony podmiotu i przedmiotu asocjacji poprzez niepodawanie referencyjnego Id w ramach samych modeli moglibyśmy utworzyć kod modeli następująco:

```
public class Entity1
{
     public Guid Id { get; set; }
     public string Label { get; set; }

     public Entity2? Entity2 { get; set; }
}
```

```
public class Entity2
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Entity1? Entity1 { get; set; }
}
```

Jednak znowu będzie to wymagało odpowiednio dłuższej konfiguracji we Fluence'sie:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Entity1>()
        .HasOne(e => e.Entity2)
        .WithOne(e => e.Entity1)
        .HasForeignKey<Entity1>("Entity2Id")
        .IsRequired(false);
    modelBuilder.Entity<Entity1>()
        .HasIndex("Entity2Id")
        .IsUnique(true);
}
```

CZĘŚĆ II Asocjacja one-to-many

1. [0-1..0-N] Asocjację taką można osiągnąć właściwie w przystępniejszy sposób niż asocjację one-to-one tj. bez konieczności dodatkowej konfiguracji przy użyciu EF Fluence i jednocześnie bez umieszczania jawnie klucza obcego w klasie modelu:

```
public class Entity1
{
     public Guid Id { get; set; }
     public string Label { get; set; }

     public List<Entity2> Entity2s { get; set; }
         = new List<Entity2>();
}
```

```
public class Entity2
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Entity1? Entity1 { get; set; }

}
```

Nullowalność deklaracji właściwości Entity1 w Entity2 sprawia, że do utworzenia obiektu modelu Entity2 nie trzeba przyporządkowywania go do żadnego obiektu Entity1.

2. [1..0-N] Podobnie jak w przypadku asocjacji [0-1..0-N] również tutaj nie ma potrzeby dodatkowej konfiguracji przy użyciu EF Fluence. Kod deklaracji modeli jest również bardzo podobny z tą różnicą że nie deklarujemy nullowalności właściwości Entity1 w Entity2:

```
public class Entity1
{
     public Guid Id { get; set; }
     public string Label { get; set; }

     public List<Entity2> Entity2s { get; set; }
         = new List<Entity2>();
}
```

```
public class Entity2
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public Entity1 Entity1 { get; set; }

}
```

CZĘŚĆ III Asocjacja many-to-many

1. [0-N..0-M] Asocjacja tego typu w bazach danych wymaga wprowadzenia dodatkowej sztucznej tabeli w której przechowywane są powiązania pomiędzy poszczególnymi obiektami obu modeli. Również tego typu asocjacje EF wspiera praktycznie przy braku

konieczności dodatkowej konfiguracji z naszej strony:

```
public class Entity1
{
     public Guid Id { get; set; }
     public string Label { get; set; }

     public List<Entity2> Entity2s{ get; set; }
         = new List<Entity2>();
}
```

```
public class Entity2
{
    public Guid Id { get; set; }
    public string Label { get; set; }

    public List<Entity1> Entity1s{ get; set; }
       = new List<Entity1>();
}
```