**Konfigurisanje klasa modela**

**00:16:40**

Jedinica: 15 od 19

**Rezime**

* Prilikom generisanja baze podataka EF Code First API se pridržava nekih ustaljenih pravila koja se nazivaju konvencije.
* Podrazumevane konvencije je moguće pregaziti konfigurisanjem domenskih klasa.
* Postoje dva načina za konfigurisanje domenskih klasa: korišćenje anotacija i FluentAPI-a.
* Unutar prostora imena System.ComponentModel.DataAnnotations nalazi se skup anotacija koje se kao atributi mogu koristiti unutar klasa modela za konfigurisanje.
* Unutar prostora imena System.ComponentModel.DataAnnotations nalaze se atributi za podešavanje null vrednosti i veličine kolona.
* FluentAPI poseduje znatno veću količinu funkcionalnosti za obavljanje konfigurisanja klasa modela.
* Da bi se konfigurisanje postiglo korišćenjem FluentAPI-a, neophodno je unutar kontekstne klase izvršiti reimplementaciju metode OnModelCreating.

U prethodnoj lekciji prikazano je kako se korišćenjem EF-a može postići Code First pristup u radu sa podacima. Jednostavno, iskorišćen je EF Code First API koji je na osnovu klasa domenskog modela vršio generisanje baze podataka. U prethodnoj lekciji takođe je prikazano kako se može uticati na naziv baze podataka i lokaciju na kojoj će ona biti kreirana, ali i koje su osnovne konvencije kojih se EF pridržava prilikom obavljanja opisanog posla. Ipak, podrazumevane konvencije je moguće pregaziti konfigurisanjem [domenskih klasa](https://www.link-elearning.com/linkdl/opisPojma.php?id=142494). Na taj način će EF dobiti informacije koje su mu potrebne za obavljanje posla.

U ovoj lekciji biće prikazano kako se može obaviti konfigurisanje klasa modela. Kao test klase poslužiće klase kreirane u prethodnoj lekciji.

Postoji dva načina za konfigurisanje domenskih klasa:

* + korišćenje anotacija,
  + FluentAPI.

**Korišćenje anotacija za konfigurisanje klasa modela**

Unutar prostora imena System.ComponentModel.DataAnnotations nalazi se skup anotacija koje se kao atributi mogu koristiti unutar klasa modela za konfigurisanje. Takvi atributi se nazivaju još i *DataAnnotations*.

Atributi iz ovog prostora imena se mogu koristiti u situacijama u kojima je potrebno postići konfigurisanje, drugačije od onog koje je podrazumevano.

Unutar prostora imena System.ComponentModel.DataAnnotations nalaze se atributi za podešavanje null vrednosti i veličine kolona. Neki od najkorišćenijih atributa iz ovog prostora imena prikazani su tabelom 15.1.

|  |  |
| --- | --- |
| **Atribut** | **Opis** |
| Key | Mapira svojstvo kao EntityKey, što rezultuje mapiranjem takvog svojstva u kolonu primarnog ključa. |
| Timestamp | Mapira svojstvo na timestamp kolonu, koja ne prihvata null vrednosti. |
| ConcurrencyCheck | Markira svojstvo kao ono koje će biti korišćeno za detekciju prilikom problema konkurentnosti. |
| Required | Osigurava da će svojstvo imati vrednost. |
| MinLength | Definiše minimalnu dužinu zapisa. |
| MaxLength | Definiše maksimalnu dužinu zapisa. |
| StringLength | Definiše maksimalni i minimalni broj karaktera zapisa. |

*Tabela 15.1 – DataAnnotations atributi*

Unutar prostora imena System.ComponentModel.DataAnnotations.Schema nalaze se atributi za podešavanje šeme baze podataka. Neki od najkorišćenijih takvih atributa su prikazani tabelom 15.2.

|  |  |
| --- | --- |
| **Atribut** | **Opis** |
| Table | Definiše naziv tabele koja će biti mapirana klasom. |
| Column | Definiše tip podatka i naziv kolone, koja će biti mapirana svojstvom. |
| Index | Kreira indeks za definisanu kolonu. |
| ForeignKey | Definiše strani ključ. |
| NotMapped | Definiše da svojstvo neće biti mapirano na bazu podataka. |
| DatabaseGenerated | Definiše da će svojstvo biti mapirano na kolonu čija se vrednost proračunava u bazi. Takve kolone mogu da budu one sa Auto Increment svojstvom ili one koje generišu vreme izvršavanja neke operacije. |
| ComplexType | Markira klasu kao kompleksni tip. |

*Tabela 15.2 – DataAnnotations.Schema atributi*

U nastavku će biti prikazan primer konfigurisanja klasa iz prethodne lekcije korišćenjem atributa.

|  |
| --- |
| ***Napomena:****za korišćenje opisanih atributa potrebno je uključiti navedene prostore imena u tekući dokument.* |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20 | [Table("PersonMaster")]  class Person  {      [Key]      public int PerId { get; set; }      [Required]      [MaxLength(50)]      [ConcurrencyCheck]      public string FirstName { get; set; }      [Required]      [StringLength(50)]      public string LastName { get; set; }      [Required]      [Column("date\_of\_birth", Order = 4, TypeName = "date")]      public DateTime DateOfBirth { get; set; }      public Country Country { get; set; }      public Profession Profession { get; set; }      [Timestamp]      public byte[] RowVersion { get; set; }  } |

U prethodnom primeru dat je primer Person klase sa definisanim atributima za konfigurisanje.

Prvi atribut koji je definisan na klasi jeste atribut Table. Podrazumevano, naziv tabele se formira po nazivu klase. Ukoliko je potrebno tabeli u bazi podataka dati neki drugi naziv, koristi se Table atribut kao u primeru. Tako će naziv generisane klase biti *PersonMaster*, a ne *Person*. Pored naziva tabele, moguće je definisati i naziv šeme u bazi podataka na sledeći način:

[Table("PersonMaster", Schema="dbo")]

Na prvom svojstvu klase Person definisan je atribut Key. S obzirom na to da naziv identifikacionog svojstva ne podleže podrazumevanim konvencijama, prilikom generisanja tabele u bazi došlo bi do greške. Na ovaj način okruženje će znati koju kolonu je potrebno da uzme kao kolonu identifikatora.

Na svojstvima *FirstName*, *LastName* i *DateOfBirth* definisan je atribut Required. Tako će EF Code First API kreirati kolone koje ne prihvataju NULL vrednosti.

Na svojstvu *FirstName* definisan je i atribut MaxLength. Ovaj atribut se može koristiti na string i array tipovima podataka. Na ovaj način će EF Code First API kreirati kolonu koja će prihvatati zapis, ne duži od 50 karaktera.

Atribut srodan atributu MaxLength je atribut StringLength. Za razliku od atributa MaxLength, on se može koristiti samo na svojstvima tipa string. Takođe se koristi za definisanje maksimalne dužine tekstualnog zapisa. Ovaj atribut je upotrebljen na svojstvu *LastName*, tako da će EF Code First API ovu kolonu u bazi kreirati sa tipom nvarchar(50).

Na svojstvu *FirstName* definisan je i atribut ConcurrencyCheck, koji može biti definisan na svojstvu bilo kojeg tipa. Na ovaj način će EF Code First API uvrstiti takvu kolonu kada se prilikom ažuriranja podataka proverava eventualni problem konkurentnosti.

Atribut TimeStamp se može primeniti samo na svojstvima koja su nizovi bajt podataka. Svojstvo sa ovakvim atributom EF Code-First API mapira na kolonu tipa timestamp, koju automatski koristi za proveru problema konkurentnosti.

Na kraju, Column atribut se može koristiti za fino podešavanje generisanja kolona. Podrazumevano, EF Code First API formira nazive kolona na osnovu naziva svojstava. Korišćenjem atributa Column, moguće je definisati naziv kolone, njen tip, kao i redni broj unutar tabele. Ovaj atribut primenjen je na koloni *DateOfBirth*. Ova kolona bi podrazumevano imala naziv *DateOfBirth*, ali je korišćenjem ove anotacije njen naziv postavljen na *date\_of\_birth*. Korišćenjem svojstva Order utiče se na redni broj kolone, a korišćenjem svojstva TypeName na tip podataka kolone.

**FluentAPI**

Drugi način za konfigurisanje domenskih klasa podrazumeva korišćenje FluentAPI-a, koji poseduje znatno veću količinu funkcionalnosti za obavljanje tog posla. Da bi se konfigurisanje postiglo korišćenjem FluentAPI-a, neophodno je unutar kontekstne klase izvršiti reimplementaciju metode OnModelCreating. Sledeći blok koda to ilustruje:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | class PersonContext : DbContext  {      public PersonContext() : base("name = TestConnString")      {      }      public DbSet<Person> People { get; set; }      public DbSet<Country> Countries { get; set; }      protected override void OnModelCreating(DbModelBuilder modelBuilder)      {          //code for configuring model classes          base.OnModelCreating(modelBuilder);      }  } |

Kompletan konfiguracioni kod FluentAPI-a smešta se unutar OnModelCreating metode. DbModelBuilder klasa je osnovna klasa koja se koristi za konfigurisanje svih domenskih tipova, a kao što se može videti, metoda OnModelCreating prihvata objekat ove klase kao parametar.

|  |
| --- |
| ***Napomena:****moguće je istovremeno korišćenje FluentAPI-a i anotacija prikazanih nešto ranije.* |

Klasa DbModelBuilder poseduje skup veoma značajnih svojstava i metoda koje se koriste za konfigurisanje domenskih klasa. U nastavku lekcije biće demonstrirane tehnike za postizanje konfigurisanja ekvivalentnog onom koje se postiže korišćenjem anotacija.

**Koja pristup omogućava veću slobodu prilikom konfigurisanja klasa modela?**

 Fluent API

 Anotacije

 Atributi

 Statičke promenljive

**Šeme**

Prvo što će biti urađeno u ovoj lekciji jeste konfigurisanje šeme za tabele baze podataka. Ovo je naravno moguće podesiti kasnije i za svaki tabelu zasebno. Sledećim blokom koda se postiže podešavanje šeme:

|  |  |
| --- | --- |
| 1  2  3  4 | protected override void OnModelCreating(DbModelBuilder modelBuilder)         {             modelBuilder.HasDefaultSchema("local");         } |

|  |
| --- |
| ***Napomena:****baza podataka je osnovni kontejner za sve podatke, log fajlove, uskladištene rutine i slično, ali i za sve šeme jedne baze. Tako se može reći da je šema logična jedinica za grupisanje podataka u okviru baze. Šeme se mogu doživeti i kao folderi unutar baze podataka.* |

**Tabele**

Da bi se definisale tabele na koje će se mapirati skupovi entiteta, koristi se metoda Entity<TEntity>(). Ovo je inače metoda kojom će započeti veći broj konfiguracionih izraza korišćenjem FluentAPI-a. Nakon metode Entity<TEntity>(), koristi se metoda ToTable() za mapiranje entiteta na tabelu. Sledeći primer to ilustruje:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | protected override void OnModelCreating(DbModelBuilder modelBuilder)  {      modelBuilder.HasDefaultSchema("local");      modelBuilder.Entity<Person>().ToTable("People");      modelBuilder.Entity<Country>().ToTable("Countries", "dbo");  } |

U primeru entitet Person je mapiran na tabelu sa nazivom People, a entitet Country na tabelu sa nazivom Countries. Prilikom mapiranja Country entiteta, metodi ToTable prosleđen je i dodatni parametar kojim se definiše šema konkretne tabele. Tako će tabela People pripadati šemi *local*, a tabela Countries šemi *dbo*.

**Primarni ključevi**

Po podrazumevanoj konvenciji, svojstvo koje će se mapirati na kolonu primarnog ključa mora imati naziv Id ili {Naziv klase} + Id. Problem nastaje kada je svojstvo potrebno imenovati na drugačiji način. U takvim situacijama se mora pregaziti podrazumevana konfiguracija. Da bi se to postiglo korišćenjem FluentAPI-a, potrebno je koristiti metodu HasKey. Sledeći primer ilustruje definisanje primarnog ključa za Person i Country entitete:

|  |  |
| --- | --- |
| 1  2  3  4  5 | protected override void OnModelCreating(DbModelBuilder modelBuilder)  {      modelBuilder.Entity<Person>().HasKey<int>(p => p.PersonKey);      modelBuilder.Entity<Country>().HasKey<int>(c => c.CountryKey);  } |

Primarni ključ može biti i kompozitni, i tada je on sačinjen od dve ili više kolona. Da bi se definisao kompozitni primarni ključ korišćenjem FluentAPI-a, može se napisati:

|  |  |
| --- | --- |
| 1  2  3  4 | protected override void OnModelCreating(DbModelBuilder modelBuilder)        {  modelBuilder.Entity<Person>().HasKey(p => new { p.PersonKey, p.FirstName });        } |

**Kolone**

Podrazumevano, EF Code First API kreira kolone na osnovu svojstava, tako da koristi identična imena, redosled i tip. Da bi se ovo promenilo, može se napisati sledeće:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | protected override void OnModelCreating(DbModelBuilder modelBuilder)  {      modelBuilder.Entity<Person>()             .Property(p => p.DateOfBirth)             .HasColumnName("date\_of\_birth")             .HasColumnOrder(3)             .HasColumnType("datetime2");  } |

Kao što se može videti u primeru, koristi se metoda Property za konfigurisanje mapiranja svojstva na kolonu. Dalje, metoda HasColumnName se koristi za promenu naziva kolone, metoda HasColumnOrder za definisanje redosleda, i metoda HasColumnType za promenu tipa podatka kolone.

**Null vrednost**

EF Code First API kreira kolone koje ne prihvataju Null vrednosti za sve primitivne podatke, pošto oni u C# programskom jeziku ne prihvataju null vrednost. Promena ovakvog ponašanja podrazumeva da se primitivni tip promeni u onaj koji prihvata null vrednosti dodavanjem karaktera ? na primitivni tip. Na taj način se dobija primitivni tip koji može da prihvati null vrednost.

Kada se koristi FluentAPI, moguće je korišćenje metoda IsOptional i IsRequired za kreiranje kolona, koje prihvataju null vrednosti, i onih koje ne prihvataju – respektivno. Sledeći primer to ilustruje:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | protected override void OnModelCreating(DbModelBuilder modelBuilder)  {      //Configure Null Column      modelBuilder.Entity<Person>()              .Property(p => p.DateOfBirth)              .IsOptional();      //Configure NotNull Column      modelBuilder.Entity<Person>()          .Property(p => p.FirstName)          .IsRequired();  } |

**Veličina kolone**

Podrazumevano, EF Code First API postavlja maksimalnu veličinu kolone za tip podatka koji kolona ima. Tako nešto je moguće promeniti korišćenjem metoda HasMaxLength, IsFixedLength iHasPrecision. Sledeći primer to ilustruje:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | protected override void OnModelCreating(DbModelBuilder modelBuilder)  {      //Set FirstName column size to 50      modelBuilder.Entity<Person>()              .Property(p => p.FirstName)              .HasMaxLength(50);      //Set FirstName column size to 50 and change datatype to nvarchar      //IsFixedLength() change datatype from nvarchar to nchar      modelBuilder.Entity<Person>()              .Property(p => p.FirstName)              .HasMaxLength(50).IsFixedLength();      //Set size decimal(2,2)      modelBuilder.Entity<Person>()          .Property(p => p.Height)          .HasPrecision(2, 2);  } |

**Konkurencija**

Da bi se neko svojstvo mapiralo na kolonu koja će se koristiti za proveru konkurentnosti korišćenjem FluentAPI-a, potrebno je koristiti metodu ConcurrencyToken, kao što je to prikazano u narednom primeru:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | protected override void OnModelCreating(DbModelBuilder modelBuilder)  {      //Set FirstName column size to 50      modelBuilder.Entity<Person>()      .Property(p => p.FirstName)      .IsConcurrencyToken();  } |

Kao što se to može videti u primeru, kolona *FirstName* je definisana kao kolona za proveru problema konkurentnosti, tako da će ona biti uvršćena u where odeljak, prilikom izvršavanja ažuriranja i brisanja.

Ekvivalentno ponašanje se može postići i metodom IsRowVersion, ali samo za svojstva koja su tipa byte[].