

Bab 6

Melampaui Dasar-Dasar dengan Pemodelan dan Inheritance

Sekarang Anda memiliki pemahaman yang kuat tentang teknik pemodelan dasar dalam Entity Framework. Dalam bab ini, Anda akan menemukan resep yang akan membantu Anda mengatasi banyak masalah pemodelan yang umum, dan seringkali rumit. Resep dalam bab ini secara khusus membahas masalah yang mungkin Anda hadapi dalam memodelkan database dunia nyata yang ada.

Kami memulai bab ini dengan bekerja dengan hubungan many-to-many. Jenis hubungan ini sangat umum dalam banyak skenario pemodelan di kedua database yang ada dan proyek-proyek baru. Selanjutnya, kita akan melihat hubungan self-referencing dan menjelajahi berbagai strategi untuk mengambil grafik objek bertingkat. Kami melengkapi bab ini dengan beberapa resep yang melibatkan model inheritance dan kondisi entitas yang lebih canggih.

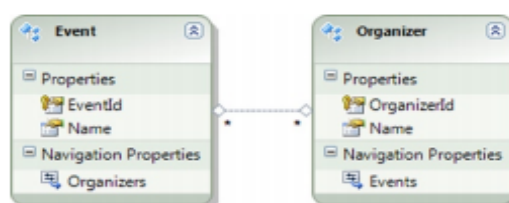
6-1. Mengambil Tabel Tautan dalam Asosiasi Many-to-Many

Masalah

Anda ingin mengambil key dalam tabel tautan yang menghubungkan dua entitas dalam asosiasi many-to-many.

Solusi

Katakanlah Anda memiliki model dengan many-to-many asosiasi antara entitas Acara dan Agenda, seperti yang diperlihatkan pada Gambar 6-1.



Gambar 6-1. Many-to-many asosiasi antara entitas Event dan Organizer

Seperti yang kami ilustrasikan dalam beberapa resep di Bab 2, hubungan many-to-many diwakili dalam database menggunakan tabel perantara yang disebut tabel tautan. Tabel tautan memegang foreign key di setiap sisi hubungan (lihat Gambar 6-2). Ketika sebuah tabel tautan tanpa tambahan kolom dan tabel terkait diimpor ke dalam Entity Framework, Entity Data Model Wizard membuat hubungan many-to-many antara tabel terkait. Tabel tautan tidak direpresentasikan sebagai entitas; namun, ini digunakan secara internal untuk asosiasi many-to-many.



Gambar 6-2. Diagram database yang memperlihatkan tabel tautan EventOrganizer yang menyimpan foreign key ke tabel Event dan Organizer terkait

Untuk mengambil kunci entitas EventId dan OrganizerId, kita dapat menggunakan metode nested dari klausa atau SelectMany (). Listing 6-1 menunjukkan kedua pendekatan.

Listing 6-1. Mengambil Tabel Tautan Menggunakan Metode Nested dari Clause dan SelectMany ()

```

using (var context = new EF6RecipesContext())
{
    var org = new Organizer { Name = "Community Charity" };
    var evt = new Event { Name = "Fundraiser" };
    org.Events.Add(evt);
    context.Organizers.Add(org);
    org = new Organizer { Name = "Boy Scouts" };
    evt = new Event { Name = "Eagle Scout Dinner" };
    org.Events.Add(evt);
    context.Organizers.Add(org);
    context.SaveChanges();
}

using (var context = new EF6RecipesContext())
{
    var evsorg1 = from ev in context.Events
    from organizer in ev.Organizers
    select new { ev.EventId, organizer.OrganizerId };
}
  
```

```

Console.WriteLine("Using nested from clauses...");
foreach (var pair in evsorg1)
{
    Console.WriteLine("EventId {0}, OrganizerId {1}",
        pair.EventId,
        pair.OrganizerId);
}

var evsorg2 = context.Events
    .SelectMany(e => e.Organizers,
        (ev, org) => new { ev.EventId, org.OrganizerId });
Console.WriteLine("\nUsing SelectMany()");

foreach (var pair in evsorg2)
{
    Console.WriteLine("EventId {0}, OrganizerId {1}",
        pair.EventId, pair.OrganizerId);
}
}

```

Output dari kode pada Listing 6-1 harus serupa dengan yang berikut:

```

Using nested from clauses...
EventId 31, OrganizerId 87
EventId 32, OrganizerId 88

Using SelectMany()
EventId 31, OrganizerId 87
EventId 32, OrganizerId 88

```

Bagaimana itu bekerja

Tabel tautan adalah cara umum mewakili hubungan many-to-many antara dua tabel dalam database. Karena tidak berfungsi selain mendefinisikan hubungan antara dua tabel, Entity Framework mewakili tabel tautan sebagai asosiasi many-to-many, bukan sebagai entitas terpisah.

Hubungan many-to-many antara Event dan Organizer memungkinkan navigasi yang mudah dari entitas Event ke organizer terkait dan dari entitas Organizer untuk semua event terkait. Namun, Anda mungkin ingin mengambilnya hanya kunci di tabel tautan. Anda mungkin ingin melakukan ini karena kunci itu sendiri bermakna atau Anda ingin menggunakan kunci ini untuk operasi pada ini atau entitas lain. Masalahnya di sini adalah bahwa tabel tautan tidak direpresentasikan sebagai entitas, jadi query langsung bukan pilihan. Dalam Listing 6-1, kami menunjukkan beberapa cara untuk mendapatkan kunci yang mendasarinya tanpa mewujudkan entitas di setiap sisi asosiasi.

Pendekatan pertama dalam Listing 6-1 menggunakan nested dari klausa untuk mengambil organizer untuk setiap event. Menggunakan properti navigasi organizer pada instance entitas event memanfaatkan tabel tautan yang mendasarinya untuk menyebutkan semua organizer untuk setiap event. Kami membentuk kembali hasil ke pasangan kunci yang sesuai untuk entitas. Akhirnya, kami melakukan iterasi melalui hasil, mencetak sepasang kunci ke konsol.

Dalam pendekatan kedua, kami menggunakan metode `SelectMany()` untuk memproyeksikan organizer untuk setiap event menjadi pasangan kunci untuk event dan organizer. Seperti halnya dengan nested dari klausa, pendekatan ini menggunakan tabel tautan yang mendasarinya melalui properti navigasi organizer. Kami melakukan iterasi melalui hasil dengan cara yang sama seperti pendekatan pertama.

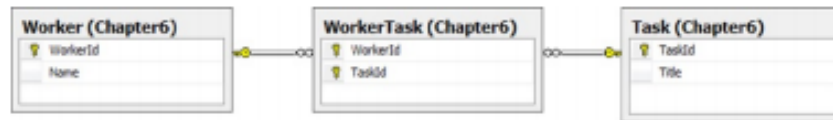
6-2. Mengekspos Tabel Tautan sebagai Entitas

Masalah

Anda ingin memaparkan tabel tautan sebagai entitas, bukan asosiasi many-to-many.

Solusi

Katakanlah bahwa database Anda memiliki hubungan many-to-many antara pekerja dan tugas, dan itu terlihat seperti yang ada di diagram database yang ditunjukkan pada Gambar 6-3.



Gambar 6-3. Hubungan many-to-many antara pekerja dan tugas

Tabel tautan WorkerTask berisi tidak lebih dari kunci asing yang mendukung hubungan many-to-many.

Untuk mengonversi asosiasi ke entitas yang mewakili tabel tautan WorkerTask, ikuti langkah-langkah ini.

1. Buat kelas entitas POCO WorkerTask, seperti yang ditunjukkan pada Listing 6-2.
2. Ganti properti Tasks dari entitas Pekerja POCO dengan properti WorkerTasks bertipe `ICollection <WorkerTask>`.
3. Ganti properti Pekerja dari entitas Task POCO dengan properti WorkerTasks dari tipe `ICollection <WorkerTask>`.
4. Tambahkan properti otomatis tipe `DbSet <WorkerTask>` ke subclass `DbContext` Anda.

Model terakhir akan terlihat seperti yang ditunjukkan pada Listing 6-2.

Listing 6-2. Model Data Akhir Termasuk WorkerTask

```

[Table("Worker", Schema="Chapter6")]
public class Worker
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int WorkerId { get; set; }
    public string Name { get; set; }
    [ForeignKey("WorkerId")]
    public virtual ICollection<WorkerTask> WorkerTasks { get; set; }
}

[Table("Task", Schema = "Chapter6")]
public class Task
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int TaskId { get; set; }
}
  
```

```

public string Title { get; set; }
[ForeignKey("TaskId")]
public virtual ICollection<WorkerTask> WorkerTasks { get; set; }
}

[Table("WorkerTask", Schema = "Chapter6")]
public class WorkerTask
{
[Key]
[Column(Order = 1)]
public int WorkerId { get; set; }
[Key]
[Column(Order = 2)]
public int TaskId { get; set; }
[ForeignKey("WorkerId")]
public virtual Worker Worker { get; set; }
[ForeignKey("TaskId")]
public virtual Task Task { get; set; }
}

```

Bagaimana itu bekerja

Selama siklus hidup pengembangan aplikasi, developer sering menemukan kebutuhan untuk menambahkan muatan ke many-to-many asosiasi yang memulai life payload-free. Dalam resep ini, kami menunjukkan cara memunculkan asosiasi many-to-many sebagai entitas terpisah sehingga properti skalar tambahan (misalnya, payload) dapat ditambahkan.

Banyak developer memilih untuk berasumsi bahwa semua hubungan many-to-many pada akhirnya akan menahan muatan, dan mereka membuat kunci sintetis untuk tabel tautan daripada kunci komposit tradisional yang dibentuk dengan menggabungkan foreign key.

Kelemahan dari model baru kami adalah kami tidak memiliki cara sederhana untuk menavigasi asosiasi many-to-many. Kami memiliki dua asosiasi one-to-many yang membutuhkan lompatan tambahan melalui entitas yang terhubung. Kode pada Listing 6-3 menunjukkan sedikit kerja tambahan pada sisi insert dan sisi query.

Listing 6-3. Memasukkan ke dalam dan Mengambil Entitas Tugas dan Pekerja

```
using (var context = new EF6RecipesContext())
{
    var worker = new Worker { Name = "Jim" };
    var task = new Task { Title = "Fold Envelopes" };
    var workertask = new WorkerTask { Task = task, Worker = worker };
    context.WorkerTasks.Add(workertask);
    task = new Task { Title = "Mail Letters" };
    workertask = new WorkerTask { Task = task, Worker = worker };
    context.WorkerTasks.Add(workertask);
    worker = new Worker { Name = "Sara" };
    task = new Task { Title = "Buy Envelopes" };
    workertask = new WorkerTask { Task = task, Worker = worker };
    context.WorkerTasks.Add(workertask);
    context.SaveChanges();
}

using (var context = new EF6RecipesContext())
{
    Console.WriteLine("Workers and Their Tasks");
    Console.WriteLine("=====");

    foreach (var worker in context.Workers)
    {
        Console.WriteLine("\n{0}'s tasks:", worker.Name);
        foreach (var wt in worker.WorkerTasks)
        {
            Console.WriteLine("\t{0}", wt.Task.Title);
        }
    }
}
```

Kode pada Listing 6-3 menghasilkan output berikut:

```
Workers and Their Tasks
=====
Jim's tasks:
    Fold Envelopes
    Mail Letters
Sara's tasks:
    Buy Envelopes
```

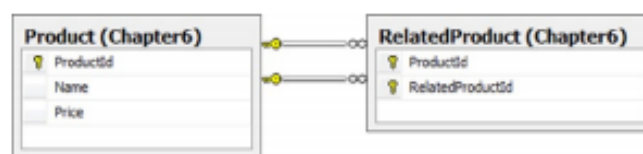
6-3. Memodelkan Hubungan Many-to-Many, Self-Referencing Relationship

Masalah

Anda memiliki table dengan hubungan many-to-many dengan dirinya sendiri, dan Anda ingin memodelkan tabel dan hubungan ini.

Solusi

Katakanlah Anda memiliki tabel yang memiliki hubungan dengan dirinya sendiri menggunakan tabel tautan, seperti yang ditunjukkan pada Gambar 6-4.



Gambar 6-4. Tabel dengan hubungan many-to-many untuk dirinya sendiri

Untuk membuat model, lakukan hal berikut:

1. Buat kelas baru di proyek Anda yang mewarisi dari DbContext.
2. Tambahkan kelas entitas POCO Produk ke proyek Anda menggunakan kode pada Listing 6-4.

Listing 6-4. Membuat Kelas Entitas POCO Produk

```
[Table("Product", Schema = "Chapter6")]
public class Product
{
    public Product()
    {
        RelatedProducts = new HashSet<Product>();
        OtherRelatedProducts = new HashSet<Product>();
    }

    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int ProductId { get; set; }
    public string Name { get; set; }
    public decimal Price { get; set; }
    // Products related to this product
    public virtual ICollection<Product> RelatedProducts { get; set; }
```



```
// Products to which this product is related
public virtual ICollection<Product> OtherRelatedProducts { get;
set; }
}
```

3. Tambahkan properti otomatis tipe DbSet <Product> ke subclass DbContext Anda.
4. Ganti metode OnModelCreating dari DbContext dalam subclass Anda untuk membuat pemetaan many-to-many self-referencing relationship, seperti yang ditunjukkan pada Listing 6-5.

Listing 6-5. Menggantikan OnModelCreating di Subclass DbContext untuk Membuat Pemetaan Many-to-Many Self-Referencing

```
protected override void OnModelCreating(DbModelBuilder
modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<Product>()
        .HasMany(p => p.RelatedProducts)
        .WithMany(p => p.OtherRelatedProducts)
        .Map(m =>
        {
            m.MapLeftKey("ProductId");
            m.MapRightKey("RelatedProductId");
            m.ToTable("RelatedProduct", "Chapter6");
        });
}
```

Bagaimana itu bekerja

Seperti yang Anda lihat, Entity Framework mendukung asosiasi many-to-many self-referencing dengan sedikit usaha. Kita menciptakan dua properti navigasi di kelas Produk kami, Produk Terkait, dan Produk Terkait Lainnya, dan dipetakan properti-properti tersebut ke skema database yang mendasari di subclass DbContext kami.

Kode pada Listing 6-6 menyisipkan beberapa produk terkait dan mengambil produk terkait. Untuk mengambil semua produk terkait untuk produk tertentu, kita perlu melintasi kedua properti navigasi RelatedProducts dan properti navigasi OtherRelatedProducts.

Tent terkait dengan Ground Cover melalui properti navigasi Produk Terkait karena kami menambahkan ground cover ke koleksi Produk Terkait Tent. Pole terkait dengan Tent melalui koleksi OtherRelatedProducts milik Tent, karena kami menambahkan koleksi yang terkait dengan Tent to Pole RelatedProducts. Asosiasi berjalan dua arah. Dalam satu arah, ini adalah produk yang terkait. Di sisi lain, ini adalah OtherRelatedProduct.

Listing 6-6. Mengambil Produk Terkait

```
using (var context = new EF6RecipesContext())
{
    var product1 = new Product { Name = "Pole", Price = 12.97M };
    var product2 = new Product { Name = "Tent", Price = 199.95M };
    var product3 = new Product { Name = "Ground Cover", Price = 29.95M
};
product2.RelatedProducts.Add(product3);
product1.RelatedProducts.Add(product2);
context.Products.Add(product1);
context.SaveChanges();
}

using (var context = new EF6RecipesContext())
{
    var product2 = context.Products.First(p => p.Name == "Tent");
    Console.WriteLine("Product: {0} ... {1}", product2.Name,
product2.Price.ToString("C"));
    Console.WriteLine("Related Products");
    foreach (var prod in product2.RelatedProducts)
    {
        Console.WriteLine("\t{0} ... {1}", prod.Name,
prod.Price.ToString("C"));
    }
    foreach (var prod in product2.OtherRelatedProducts)
    {
        Console.WriteLine("\t{0} ... {1}", prod.Name,
prod.Price.ToString("C"));
    }
}
```

Output dari Listing 6-6 adalah sebagai berikut:

```
Product: Tent ... $199.95
Related Products
  Ground Cover ... $29.95
  Pole ... $12.97
```

Kode pada Listing 6-6 hanya mengambil level pertama dari produk terkait. Hubungan transitif adalah hubungan yang mencakup beberapa level, seperti hierarki. Jika kita berasumsi bahwa hubungan "produk terkait" bersifat transitif, kita mungkin ingin membentuk penutupan transitif. Penutupan transitif adalah semua produk yang terkait terlepas dari berapa banyak lompatannya. Dalam aplikasi eCommerce, spesialis produk dapat membuat tingkat pertama produk terkait. Tingkat tambahan dapat diturunkan dengan menghitung penutupan transitif. Hasil akhirnya akan memungkinkan aplikasi untuk menampilkan pesan "... Anda juga mungkin tertarik..." yang sering kita lihat selama proses checkout.

Dalam Listing 6-7, kami menggunakan metode rekursif untuk membentuk penutupan transitif. Dalam melintasi asosiasi Produk Terkait dan OtherRelatedProducts, kita harus berhati-hati agar tidak terjebak dalam siklus. Jika produk A terkait dengan B, dan produk B terkait dengan produk A, aplikasi kita akan terjebak dalam rekursi. Untuk mendeteksi siklus, kami menggunakan Kamus <> untuk membantu memangkas jalur yang telah kami lalui.

Listing 6-7. Membentuk Penutupan Transitif dari Hubungan "Produk Terkait"

```
static void RunExample2()
{
    using (var context = new EF6RecipesContext())
    {
        var product1 = new Product { Name = "Pole", Price = 12.97M };
        var product2 = new Product { Name = "Tent", Price = 199.95M };
        var product3 = new Product { Name = "Ground Cover", Price = 29.95M };

        product2.RelatedProducts.Add(product3);
        product1.RelatedProducts.Add(product2);
        context.Products.Add(product1);
        context.SaveChanges();
    }
    using (var context = new EF6RecipesContext())
    {
```

```

var product1 = context.Products.First(p => p.Name == "Pole");
Dictionary<int, Product> t = new Dictionary<int, Product>();
GetRelated(context, product1, t);
Console.WriteLine("Products related to {0}", product1.Name);
foreach (var key in t.Keys)
{
    Console.WriteLine("\t{0}", t[key].Name);
}
}
}

static void GetRelated(DbContext context, Product p, Dictionary<int,
Product> t)
{
    context.Entry(p).Collection(ep => ep.RelatedProducts).Load();
    foreach (var relatedProduct in p.RelatedProducts)
    {
        if (!t.ContainsKey(relatedProduct.ProductId))
        {
            t.Add(relatedProduct.ProductId, relatedProduct);
            GetRelated(context, relatedProduct, t);
        }
    }
    context.Entry(p).Collection(ep => ep.OtherRelatedProducts).Load();
    foreach (var otherRelated in p.OtherRelatedProducts)
    {
        if (!t.ContainsKey(otherRelated.ProductId))
        {
            t.Add(otherRelated.ProductId, otherRelated);
            GetRelated(context, otherRelated, t);
        }
    }
}
}

```

Dalam Listing 6-7, kami menggunakan metode Load () (lihat resep di Bab 5) untuk memastikan bahwa koleksi produk terkait dimuat. Sayangnya, ini berarti bahwa kita akan berakhir dengan banyak perjalanan tambahan ke database. Kami mungkin tergoda untuk memuat semua baris dari tabel Produk di muka dan berharap rentang hubungan akan

memperbaiki asosiasi. Namun, rentang hubungan tidak akan memperbaiki koleksi entitas, hanya referensi entitas. Karena asosiasi kami many-to-many (koleksi entitas), kami tidak dapat mengandalkan rentang hubungan untuk membantu dan kami harus menggunakan metode Load ().

Berikut ini adalah hasil dari kode pada Listing 6-7. Dari blok kode pertama yang menyisipkan hubungan, kita dapat melihat bahwa Pole terkait dengan Tent, dan Tent terkait dengan Ground Cover. Penutupan transitif untuk produk yang terkait dengan Pole termasuk Tent, Ground Cover, dan Pole. Pole disertakan karena berada di sisi lain dari hubungan dengan Tent, yang merupakan produk terkait.

```
Products related to Pole
  Tent
  Ground Cover
  Pole
```

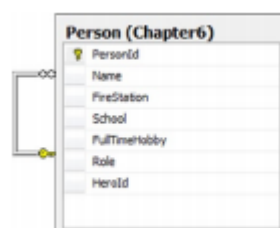
6-4. Memodelkan Hubungan Self-Referencing Menggunakan Tabel per Hierarki Inheritance

Masalah

Anda memiliki tabel yang merujuk sendiri. Tabel mewakili beberapa jenis objek yang berbeda tetapi terkait dalam database Anda. Anda ingin memodelkan tabel ini menggunakan Table per Hierarchy inheritance.

Solusi

Misalkan Anda memiliki tabel seperti yang ditunjukkan pada Gambar 6-5, yang menggambarkan beberapa hal tentang orang. Orang sering memiliki pahlawan, mungkin individu yang paling mengilhami mereka. Kami dapat mewakili pahlawan seseorang dengan referensi ke baris lain di tabel Orang.



Gambar 6-5. Tabel orang yang berisi orang-orang dengan peran berbeda

Setiap orang memiliki beberapa peran dalam kehidupan. Beberapa orang adalah petugas pemadam kebakaran. Sebagian orang adalah guru. Sebagian orang sudah pensiun. Tentu saja, ada banyak peran lain. Informasi tentang orang bisa spesifik untuk peran mereka. Seorang petugas pemadam kebakaran ditempatkan di sebuah pos pemadam kebakaran. Seorang guru mengajar di sekolah. Pensiunan sering memiliki hobi.

Untuk contoh kita, peran yang mungkin adalah petugas pemadam kebakaran (f), guru (t), atau pensiunan (r). Satu karakter dalam kolom peran menunjukkan peran untuk seseorang.

Untuk membuat model, lakukan hal berikut:

1. Buat kelas baru di proyek Anda yang mewarisi dari DbContext
2. Tambahkan kelas entitas Person POCO abstrak menggunakan kode yang ditemukan pada Listing 6-8.

Listing 6-8. Menciptakan Kelas Entitas POCO Abstrak Person

```
[Table("Person", Schema = "Chapter6")]
public abstract class Person
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int PersonId { get; protected set; }
    public string Name { get; set; }
    public virtual Person Hero { get; set; }
    public virtual ICollection<Person> Fans { get; set; }
}
```

3. Tambahkan properti otomatis tipe DbSet <Person> ke subclass DbContext Anda.
4. Tambahkan kelas entitas concrete POCO untuk entitas Pemadam Kebakaran, Guru, dan Pensiunan menggunakan kode yang ditemukan pada Listing 6-9.

Listing 6-9. Menciptakan Entitas concrete POCO untuk Pemadam Kebakaran, Guru, dan Pensiunan

```
public class Firefighter : Person
{
    public string FireStation { get; set; }
}

public class Teacher : Person
```

```

{
public string School { get; set; }
}
public class Retired : Person
{
public string FullTimeHobby { get; set; }
}

```

5. Ganti metode `OnModelCreating` dari `DbContext` di subkelas Anda untuk mengonfigurasi foreign key Herold serta hierarki tipe, seperti yang ditunjukkan pada Listing 6-10.

Listing 6-10. Overriding `OnModelCreating` dalam `DbContext` Subclass

```

protected override void OnModelCreating(DbModelBuilder
modelBuilder)
{
base.OnModelCreating(modelBuilder);
modelBuilder.Entity<Person>()

.HasMany(p => p.Fans)
.WithOptional(p => p.Hero)
.Map(m => m.MapKey("HeroId"));
modelBuilder.Entity<Person>()
.Map<Firefighter>(m => m.Requires("Role").HasValue("f"))
.Map<Teacher>(m => m.Requires("Role").HasValue("t"))
.Map<Retired>(m => m.Requires("Role").HasValue("r"));
}

```

Bagaimana itu bekerja

Kode pada Listing 6-11 menunjukkan memasukkan dan mengambil entitas `Person` dari model kami. Kami membuat satu contoh dari masing-masing jenis dan wire turunan dalam beberapa hubungan pahlawan. Kami memiliki seorang guru yang merupakan pahlawan dari seorang petugas pemadam kebakaran dan seorang pensiunan yang merupakan pahlawan dari guru. Ketika kami menetapkan pemadam kebakaran sebagai pahlawan dari pensiunan, kami cukup memperkenalkan satu siklus sehingga Entity Framework menghasilkan kesalahan runtime (`DbUpdateException`) karena tidak dapat menentukan urutan yang tepat untuk memasukkan

baris ke dalam tabel. Dalam kode, kita mengatasi masalah ini dengan memanggil metode `SaveChanges ()` sebelum memasang wire di salah satu hubungan pahlawan. Setelah baris berkomitmen ke database, dan store-generated keys dibawa kembali ke grafik objek, kami bebas memperbarui grafik dengan hubungan. Tentu saja, perubahan ini harus disimpan dengan panggilan akhir ke `SaveChanges ()`.

Listing 6-11. Memasukkan ke dalam dan Mengambil dari Model Kami

```
using (var context = new EF6RecipesContext())
{
    var teacher = new Teacher { Name = "Susan Smith",
        School = "Custer Baker Middle School" };
    var firefighter = new Firefighter { Name = "Joel Clark",
        FireStation = "Midtown" };
    var retired = new Retired { Name = "Joan Collins",
        FullTimeHobby = "Scapbooking" };
    context.People.Add(teacher);
    context.People.Add firefighter);
    context.People.Add(retired);
    context.SaveChanges();
    firefighter.Hero = teacher;
    teacher.Hero = retired;
    retired.Hero = firefighter;
    context.SaveChanges();
}

using (var context = new EF6RecipesContext())
{
    foreach (var person in context.People)
    {
        if (person.Hero != null)
            Console.WriteLine("\n{0}, Hero is: {1}", person.Name,
                person.Hero.Name);
        else
            Console.WriteLine("{0}", person.Name);

        if (person is Firefighter)
            Console.WriteLine("Firefighter at station {0}",
                ((Firefighter)person).FireStation);
    }
}
```



```

else if (person is Teacher)
Console.WriteLine("Teacher at {0}", ((Teacher)person).School);
else if (person is Retired)
Console.WriteLine("Retired, hobby is {0}",
((Retired)person).FullTimeHobby);
Console.WriteLine("Fans:");
foreach (var fan in person.Fans)
{
Console.WriteLine("\t{0}", fan.Name);
}
}
}
}

```

Output dari kode pada Listing 6-11 adalah sebagai berikut:

```

Susan Smith, Hero is: Joan Collins
Teacher at Custer Baker Middle School
Fans:
    Joel Clark

Joel Clark, Hero is: Susan Smith
Firefighter at station Midtown
Fans:
    Joan Collins

Joan Collins, Hero is: Joel Clark
Retired, hobby is Scapbooking
Fans:
    Susan Smith

```

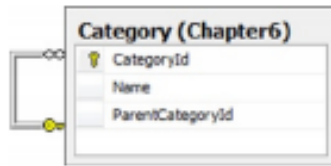
6-5. Memodelkan Hubungan Self-Referencing dan Mengambil Hirarki Lengkap

Masalah

Anda menggunakan tabel self-referencing untuk menyimpan data hierarkis. Dengan catatan, Anda ingin mengambil semua rekaman terkait yang merupakan bagian dari hierarki tersebut pada level mana pun.

Solusi

Misalkan Anda memiliki tabel Kategori seperti yang ada dalam diagram database yang ditunjukkan pada Gambar 6-6.



Gambar 6-6. Tabel Kategori self-referencing

Untuk membuat model kami, lakukan hal berikut:

1. Buat kelas baru di proyek Anda yang mewarisi dari DbContext.
2. Tambahkan kelas entitas Kategori POCO menggunakan kode pada Listing 6-12.

Listing 6-12. Membuat Kategori Kelas Entity POCO

```
[Table("Category", Schema = "Chapter6")]
public class Category
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int CategoryId { get; set; }
    public string Name { get; set; }
    public virtual Category ParentCategory { get; set; }
    public virtual ICollection<Category> SubCategories { get; set; }
}
```

3. Tambahkan properti otomatis tipe DbSet <Kategori> ke subkelas DbContext.
4. Ganti metode OnModelCreating dari DbContext dalam subclass Anda, seperti yang ditunjukkan pada Listing 6-13. Dalam penggantian, kita akan membuat asosiasi ParentCategory dan SubCategories dan mengkonfigurasi batasan foreign key.

Listing 6-13. Overriding OnModelCreating dalam DbContext Subclass

```
protected override void OnModelCreating(DbModelBuilder
modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<Category>()
        .HasOptional(c => c.ParentCategory)
        .WithMany(c => c.SubCategories)
        .Map(m => m.MapKey("ParentCategoryId"));
}
```

Dalam model kami, entitas Kategori memiliki properti navigasi Subkategori yang dapat kami gunakan untuk mendapatkan koleksi semua subkategori langsung dari Kategori. Namun, untuk mengaksesnya, kita perlu memuatnya secara eksplisit menggunakan metode Load () atau Include (). Metode Load () membutuhkan tambahan round trip ke database, sedangkan method Include () hanya menyediakan kedalaman yang telah ditentukan, terbatas.

Kami ingin membawa seluruh hirarki ke dalam grafik objek seefisien mungkin. Untuk melakukan ini, kami menggunakan Ekspresi Tabel Umum dalam prosedur yang tersimpan.

Untuk menambahkan prosedur yang tersimpan ke model kami, lakukan hal berikut:

5. Buat prosedur tersimpan yang disebut GetSubCategories yang memanfaatkan Ekspresi Tabel Umum untuk mengembalikan semua subkategori untuk CategoryId secara rekursif. Prosedur tersimpan ditunjukkan pada Listing 6-14.

Listing 6-14. GetSubCategories () Stored Procedure Itu Mengembalikan Subkategori untuk CategoryId yang Diberikan

```
create proc chapter6.GetSubCategories
(@categoryid int)
as
begin
with cats as
(
select c1.*
from chapter6.Category c1
where CategoryId = @categoryid
union all
select c2.*
from cats join chapter6.Category c2 on cats.CategoryId =
c2.ParentCategoryId
)
select * from cats where CategoryId != @categoryid
end
```

6. Tambahkan metode yang mengambil parameter integer dan mngembalikan ICollection <Category> ke subclass DbContext Anda, seperti yang ditunjukkan pada Listing 6-15. Entity Framework 6 Kode Pertama belum mendukung impor fungsi seperti yang dilakukan oleh perancang EF, jadi dalam tubuh metode kami akan memanggil prosedur tersimpan kami dengan metode SqlQuery yang didefinisikan dalam properti Database DbContext.

Listing 6-15. Menerapkan Metode GetSubCategories dalam Subclass DbContext kami

```
public ICollection<Category> GetSubCategories(int categoryId)
{
    return this.Database.SqlQuery<Category>("exec
Chapter6.GetSubCategories @catId",
new SqlParameter("@catId", categoryId)).ToList();
}
```

Kita dapat menggunakan metode GetSubCategories yang telah kita definisikan dalam subkelas DbContext kami untuk mewujudkan seluruh grafik kategori dan subkategori kami. Kode dalam Listing 6-16 menunjukkan penggunaan metode GetSubCategories ().

Listing 6-16. Mengambil Seluruh Hirarki Menggunakan Metode GetSubCategories ()

```
using (var context = new EF6RecipesContext())
{
    var book = new Category { Name = "Books" };
    var fiction = new Category { Name = "Fiction", ParentCategory = book
};
    var nonfiction = new Category { Name = "Non-Fiction", ParentCategory
= book };
    var novel = new Category { Name = "Novel", ParentCategory = fiction
};
    var history = new Category { Name = "History", ParentCategory =
nonfiction };

    context.Categories.Add(novel);
    context.Categories.Add(history);
    context.SaveChanges();
}
using (var context = new EF6RecipesContext())
{
```

```

var root = context.Categories.Where(o => o.Name == "Books").First();
Console.WriteLine("Parent category is {0}, subcategories are:",
root.Name);
foreach (var sub in context.GetSubCategories(root.CategoryId))
{
    Console.WriteLine("\t{0}", sub.Name);
}
}

```

Output dari kode dalam Listing 6-16 adalah sebagai berikut:

```

Parent category is Books, subcategories are:
    Fiction
    Non-Fiction
    History
    Novel

```

Bagaimana itu bekerja

Entity Framework mendukung asosiasi self-referencing, seperti yang telah kita lihat di Resep 6.2 dan Resep 6.3. Dalam resep ini, kami langsung memuat referensi entitas dan koleksi menggunakan metode `Load()`. Kami memperingatkan, bagaimanapun, bahwa setiap `Load()` menghasilkan perjalanan bolak-balik ke database untuk mengambil entitas atau kumpulan entitas. Untuk grafik objek yang lebih besar, lalu lintas database ini dapat menghabiskan terlalu banyak sumber daya.

Dalam resep ini, kami menunjukkan pendekatan yang sedikit berbeda. Daripada secara eksplisit menggunakan `Load()` untuk mewujudkan setiap entitas atau kumpulan entitas, kami mendorong pekerjaan ke lapisan penyimpanan dengan menggunakan prosedur yang tersimpan untuk menyebutkan secara rekursif semua subkategori dan mengembalikan koleksi. Kami menggunakan Ekspresi Tabel Umum dalam prosedur tersimpan kami untuk menerapkan permintaan rekursif. Dalam contoh kami, kami memilih untuk menyebutkan semua subkategori. Anda bisa, tentu saja, memodifikasi prosedur yang tersimpan untuk menyebutkan elemen hierarki secara selektif.

Untuk menggunakan prosedur tersimpan kami, kami menambahkan metode untuk subclass `DbContext` kami yang memanggil prosedur yang tersimpan melalui `DbContext.Database.SqlQuery <T> ()` dan memanggil metode dalam kode kami. Kami menggunakan metode `SqlQuery <T> ()` daripada metode `ExecuteSqlCommand ()` karena prosedur tersimpan kami mengembalikan set hasil.

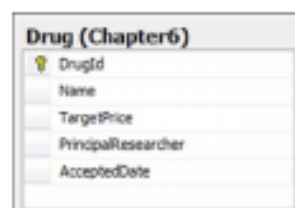
6-6. Memetakan Kondisi Null di Entitas Derived

Masalah

Anda memiliki kolom dalam tabel yang memungkinkan nol. Anda ingin membuat model menggunakan Tabel per Hirarki inheritance dengan satu jenis turunan yang mewakili contoh di mana kolom memiliki nilai dan jenis turunan lain yang mewakili contoh di mana kolom tidak valid.

Solusi

Katakanlah Anda memiliki tabel yang menjelaskan obat medis eksperimental. Tabel berisi kolom yang menunjukkan kapan obat itu diterima untuk produksi. Sampai obat diterima untuk produksi, itu dianggap eksperimental. Setelah diterima, itu dianggap sebagai obat. Kami akan mulai dengan tabel Obat dalam diagram database pada Gambar 6-7.



Gambar 6-7. Tabel obat dengan kolom diskriminator nullable, `AcceptedDate`

Untuk membuat model menggunakan tabel Obat, lakukan hal berikut:

1. Buat kelas di proyek Anda yang mewarisi dari `DbContext`.
2. Buat kelas `Drug`, `Medicine`, dan `Eksperimental POCO`, seperti yang ditunjukkan pada Listing 6-17.

Listing 6-17. Membuat Kelas Entitas Drug, Medicine, dan Eksperimental POCO

```
[Table("Drug", Schema = "Chapter6")]
public abstract class Drug
{
    [Key]
    [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
    public int DrugId { get; set; }
    public string Name { get; set; }
}

public class Experimental : Drug
{
    public string PrincipalResearcher { get; set; }
    public void PromoteToMedicine(DateTime acceptedDate, decimal
    targetPrice,
    string marketingName)
    {
        var drug = new Medicine { DrugId = this.DrugId };
        using (var context = new DrugContext())
        {
            context.Drugs.Attach(drug);
            drug.AcceptedDate = acceptedDate;
            drug.TargetPrice = targetPrice;
            drug.Name = marketingName;
            context.SaveChanges();
        }
    }
}

public class Medicine : Drug
{
    public decimal? TargetPrice { get; set; }
    public DateTime AcceptedDate { get; set; }
}
```

3. Tambahkan properti otomatis tipe `DbSet <Drug>` ke subkelas `DbContext` Anda.
4. Ganti metode `OnModelCreating` dari `DbContext` untuk mengonfigurasi pemetaan TPH untuk jenis `Medicine` dan `Eksperimental`, seperti yang ditunjukkan pada Listing 6-18.

Listing 6-18. Overriding OnModelCreating untuk Mengonfigurasi Pemetaan TPH

```
protected override void OnModelCreating(DbModelBuilder
modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<Experimental>()
        .Map(m => m.Requires("AcceptedDate").HasValue((DateTime?) null));
    modelBuilder.Entity<Medicine>()
        .Map(m => m.Requires(d => d.AcceptedDate).HasValue());
}
```

Bagaimana itu bekerja

Dalam contoh ini, kami menggunakan kondisi null dan bukan null untuk memetakan Obat tanpa AcceptedDate ke obat Eksperimental dan Obat dengan AcceptedDate to a Medicine. Seperti dalam banyak contoh inheritance, kami menandai entitas dasar, Obat, sebagai abstrak karena dalam model kami, kami tidak akan pernah memiliki obat yang tidak dikategorikan.

Sangat menarik untuk dicatat bahwa, dalam entitas Medicine, kami memetakan kolom discriminated AcceptedDate ke properti skalar. Dalam kebanyakan skenario, memetakan kolom diskriminator ke properti skalar dilarang. Namun, dalam contoh ini, kami menggunakan kondisi null dan bukan null, serta menandai AcceptedDate sebagai tidak nullable, cukup membatasi nilai properti untuk memungkinkan pemetaan.

Dalam Listing 6-19, kami memasukkan beberapa obat Eksperimental dan menanyakan hasilnya. Kami mengambil kesempatan yang disediakan oleh properti AcceptedDate terbuka untuk mendemonstrasikan salah satu cara untuk mengubah objek dari satu jenis turunan ke yang lain. Dalam kasus kami, kami membuat beberapa obat Eksperimental dan kemudian mempromosikan salah satunya ke Medicine.

Listing 6-19. Memasukkan dan Mengambil Instance dari Tipe Berasal Kami

```
class Program
{
    ...
    static void RunExample()
    {
        using (var context = new EF6RecipesContext())
        {
            var exDrug1 = new Experimental { Name = "Nanoxol",
            PrincipalResearcher = "Dr. Susan James" };
            var exDrug2 = new Experimental { Name = "Percosol",
            PrincipalResearcher = "Dr. Bill Minor" };
            context.Drugs.Add(exDrug1);
            context.Drugs.Add(exDrug2);
            context.SaveChanges();

            // Nanoxol just got approved!
            exDrug1.PromoteToMedicine(DateTime.Now, 19.99M, "Treatall");
            context.Entry(exDrug1).State = EntityState.Detached // better not
            use this instance any longer
        }
        using (var context = new EF6RecipesContext())
        {
            Console.WriteLine("Experimental Drugs");
            foreach (var d in context.Drugs.OfType<Experimental>())
            {
                Console.WriteLine("\t{0} ({1})", d.Name, d.PrincipalResearcher);
            }
            Console.WriteLine("Medicines");
            foreach (var d in context.Drugs.OfType<Medicine>())
            {
                Console.WriteLine("\t{0} Retails for {1}", d.Name,
                d.TargetPrice.Value.ToString("C"));
            }
        }
    }
}
```

Berikut ini adalah hasil dari kode pada Listing 6-19:

```
Experimental Drugs
  Percosol (Dr. Bill Minor)
Medicines
  Treatall Retails for $19.99
```

Kami mengubah obat Eksperimen ke Medicine menggunakan metode `PromoteToMedicine()`. Dalam penerapan metode ini, kami membuat instance Obat baru, melampirkannya ke `DbContext` baru, dan menginisialisasi dengan nilai baru yang sesuai. Setelah instance baru dilampirkan dan diinisialisasi, kami menggunakan metode `SaveChanges()` pada `DbContext` untuk menyimpan instance baru ke database. Karena instance memiliki kunci yang sama (`DrugId`) sebagai obat Eksperimen, Entity Framework menghasilkan pernyataan pembaruan daripada pernyataan sisipan.

Kami menerapkan metode `PromoteToMedicine()` di dalam `Eksperimental` kelas `POCO`. Ini memungkinkan kami untuk menambahkan metode ke kelas dengan lancar, dan ini memberikan implementasi yang lebih bersih. Dikatakan demikian, demi kepentingan menciptakan entitas `POCO` persistence-ignorant yang dapat digunakan dalam beberapa `DbContext`, mungkin lebih masuk akal untuk menerapkan versi yang sedikit diubah dari metode ini dalam kelas `instead`.

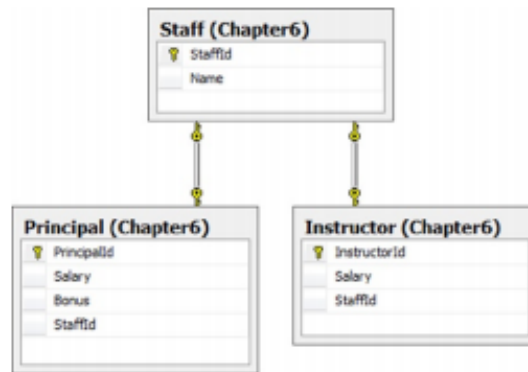
6-7. Tabel Pemodelan per Tipe Inheritance Menggunakan Kolom Nonprimary Key

Masalah

Anda memiliki satu atau beberapa tabel dalam skema yang ada yang memiliki hubungan one-to-one ke tabel umum menggunakan kunci yang bukan primary key dalam tabel. Anda ingin memodelkan ini menggunakan Table per Type inheritance.

Solusi

Misalnya, database Anda berisi tabel yang ditampilkan dalam diagram database pada Gambar 6-8.



Gambar 6-8. Diagram database yang berisi tabel Staf, Kepala Sekolah, dan Instruktur

Pada Gambar 6-8, kami memiliki tabel Staf yang berisi nama anggota staf dan dua tabel terkait yang berisi informasi tentang Kepala Sekolah dan Instruktur. Hal yang penting untuk diperhatikan di sini adalah bahwa tabel Pokok dan Instruktur memiliki primary key yang bukan foreign key untuk tabel Staf. Jenis struktur hubungan ini tidak didukung secara langsung di Tabel per Tipe Inheritance. Untuk Tabel per Tipe, primary key tabel terkait juga harus merupakan foreign key untuk tabel utama (dasar). Juga perhatikan bahwa hubungannya adalah one-to-one. Ini karena kami telah membatasi kolom StaffId di tabel Utama dan Instruktur menjadi unik dengan membuat indeks unik pada kolom ini di kedua tabel.

Untuk memodelkan tabel dan hubungan dalam Gambar 6-8 menggunakan Tabel per TipeInheritance, lakukan hal berikut:

1. Tambahkan Model Data Entitas ADO.NET baru ke proyek Anda, dan impor tabel Staf, Kepala Sekolah, dan Instruktur.
2. Hapus asosiasi antara Kepala Sekolah dan entitas Staf dan antara Instruktur dan entitas Staf.
3. Klik kanan entitas Staf, dan pilih Add ► Inheritance. Pilih Staf sebagai entitas dasar dan Prinsipal sebagai entitas turunan. Ulangi langkah ini dengan memilih Staf sebagai entitas dasar dan Instruktur sebagai entitas turunan.
4. Hapus properti StaffId dari Instruktur dan entitas Utama.
5. Klik kanan entitas Staff, dan pilih Properties. Set atribut Abstrak ke True. Ini menandai entitas Staff sebagai abstrak.

6. Karena StaffId bukan primary key baik dalam tabel Principal atau Instruktur, kita tidak dapat menggunakan pemetaan tabel default untuk memetakan entitas Principal, Instruktur, atau Staf. Pilih setiap entitas, lihat jendela Detail Pemetaan, dan hapus pemetaan tabel. Ulangi ini untuk setiap entitas.
7. Buat prosedur tersimpan dalam Listing 6-20. Kami akan memetakan prosedur ini ke action Insert, Update, dan Delete untuk entitas Principal dan Instruktur.

Listing 6-20. Stored Procedures untuk action Insert, Update, dan Delete untuk Instruktur dan Entitas Utama

```
create procedure [chapter6].[InsertInstructor]
(@Name varchar(50), @Salary decimal)
as
begin
declare @staffid int
insert into Chapter6.Staff(Name) values (@Name)
set @staffid = SCOPE_IDENTITY()
insert into Chapter6.Instructor(Salary,StaffId) values
(@Salary,@staffid)
select @staffid as StaffId,SCOPE_IDENTITY() as InstructorId
end
go

create procedure [chapter6].[UpdateInstructor]
(@Name varchar(50), @Salary decimal, @StaffId int, @InstructorId
int)
as
begin
update Chapter6.Staff set Name = @Name where StaffId = @StaffId
update Chapter6.Instructor set Salary = @Salary where
InstructorId = @InstructorId
end
go

create procedure [chapter6].[DeleteInstructor]
(@StaffId int)
as
begin
delete Chapter6.Staff where StaffId = @StaffId
delete Chapter6.Instructor where StaffId = @StaffId
```

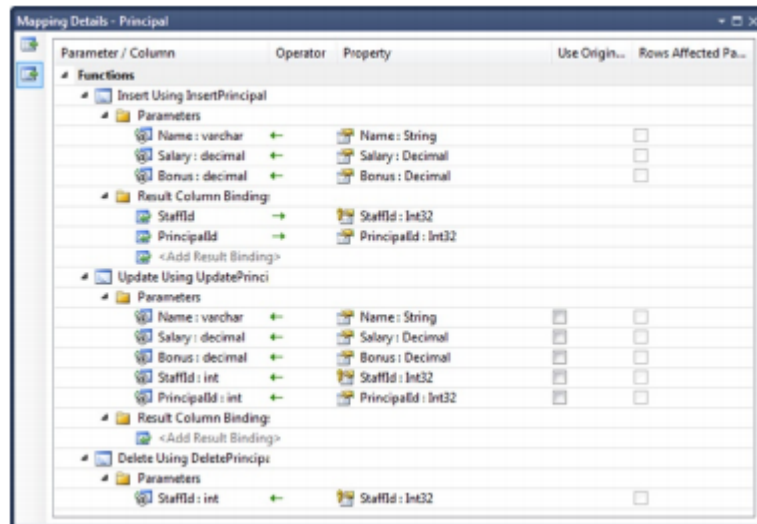
```

end
go
create procedure [Chapter6].[InsertPrincipal]
(@Name varchar(50),@Salary decimal,@Bonus decimal)
as
begin
declare @staffid int
insert into Chapter6.Staff(Name) values (@Name)
set @staffid = SCOPE_IDENTITY()
insert into Chapter6.Principal(Salary,Bonus,StaffId) values
(@Salary,@Bonus,@staffid)
select @staffid as StaffId, SCOPE_IDENTITY() as PrincipalId
end
go
create procedure [Chapter6].[UpdatePrincipal]
(@Name varchar(50),@Salary decimal, @Bonus decimal, @StaffId int,
@PrincipalId int)
as
begin
update Chapter6.Staff set Name = @Name where StaffId = @StaffId
update Chapter6.Principal set Salary = @Salary, Bonus = @Bonus
where
PrincipalId = @PrincipalId
end
go
create procedure [Chapter6].[DeletePrincipal]
(@StaffId int)
as
begin
delete Chapter6.Staff where StaffId = @StaffId
delete Chapter6.Principal where StaffId = @StaffId
end

```

8. Klik kanan permukaan desain, dan pilih Perbarui Model dari Database. Tambahkan prosedur tersimpan yang Anda buat pada langkah 7.

9. Pilih entitas Utama, dan lihat jendela Detail Pemetaan. Klik tombol Map Entity to Functions. Ini adalah tombol bawah di sisi kiri jendela Rincian Pemetaan. Petakan tindakan Insert, Update, dan Delete ke prosedur tersimpan. Pastikan bahwa Anda memetakan kolom hasil StaffId dan PrincipalId dari tindakan Insert (lihat Gambar 6-9).



Gambar 6-9. Action Insert, Update, dan Delete yang dipetakan untuk entitas Utama

10. Ulangi langkah 9 untuk entitas Instruktur. Pastikan untuk memetakan kolom hasil StaffId dan InstructorId dari tindakan Insert.

Klik kanan file .edmx di Solution Explorer, dan pilih Open With ► Editor XML. Ini akan menutup perancang dan membuka file .edmx di editor XML. Gulir ke bawah ke <EntityContainerMapping> tag di lapisan pemetaan. Masukkan QueryView di Listing 6-21 ke dalam tag <EntitySetMapping>.

Listing 6-21. QueryView untuk Instruktur dan Entitas Utama

```
<EntitySetMapping Name="Staffs">
  <QueryView>
    select value
    case
    when (i.StaffId is not null) then
      EFRecipesModel.Instructor(s.StaffId,s.Name,i.InstructorId,i.Salary)
    when (p.StaffId is not null) then
      EFRecipesModel.Principal(s.StaffId,s.Name,p.PrincipalId,p.Salary,p.B
      onus)
    END
  from EFRecipesModelStoreContainer.Staff as s
```

```
left join EFRecipesModelStoreContainer.Instructor as i
on s.StaffId = i.StaffId
left join EFRecipesModelStoreContainer.Principal as p
on s.StaffId = p.StaffId
</QueryView>
</EntitySetMapping>
```

Bagaimana itu bekerja

Dengan Table per Type inheritance, Entity Framework mengharuskan foreign key untuk tabel entitas dasar menjadi primary key dalam tabel entitas turunan. Dalam contoh kami, masing-masing tabel untuk entitas turunan memiliki primary key terpisah.

Untuk membuat model Table per Type inheritance, kami memulai pada level konseptual dengan menurunkan entitas Principal dan Instruktur dari entitas Staff. Selanjutnya kami menghapus pemetaan yang dibuat saat kami mengimpor tabel. Kami kemudian menggunakan ekspresi QueryView untuk membuat pemetaan baru. Menggunakan QueryView mendorong tanggung jawab untuk action Insert, Update, dan Delete ke kode kami. Untuk menangani action ini, kami menggunakan prosedur tersimpan tradisional dalam database.

Kami menggunakan QueryView untuk menyediakan pemetaan dari tabel yang mendasari kami ke properti skalar yang diekspos oleh entitas turunan kami. Bagian kunci dari QueryView adalah pernyataan kasus. Ada dua kasus: apakah kami memiliki Kepala Sekolah atau kami memiliki seorang Instruktur. Kami memiliki Instruktur jika Staf Pengajar tidak batal, atau kami memiliki Kepala Sekolah jika StaffId Kepala Sekolah tidak batal. Bagian-bagian lain dari ekspresi membawa baris dari tabel turunan.

Kode dalam Listing 6-22 memasukkan beberapa Prinsip dan satu Instruktur ke dalam database kami.

Listing 6-22. Memasukkan ke dalam dan Mengambil dari Model Kami

```
using (var context = new EF6RecipesContext())
{
    var principal = new Principal { Name = "Robbie Smith",
    Bonus = 3500M, Salary = 48000M };
    var instructor = new Instructor { Name = "Joan Carlson",
    Salary = 39000M };

    context.Staffs.Add(principal);
    context.Staffs.Add(instructor);
    context.SaveChanges();
}

using (var context = new EF6RecipesContext())
{
    Console.WriteLine("Principals");
    Console.WriteLine("=====");
    foreach (var p in context.Staffs.OfType<Principal>())
    {
        Console.WriteLine("\t{0}, Salary: {1}, Bonus: {2}",
        p.Name, p.Salary.ToString("C"),
        p.Bonus.ToString("C"));
    }
    Console.WriteLine("Instructors");
    Console.WriteLine("=====");
    foreach (var i in context.Staffs.OfType<Instructor>())
    {
        Console.WriteLine("\t{0}, Salary: {1}", i.Name,
        i.Salary.ToString("C"));
    }
}
```

Berikut ini adalah hasil dari kode pada Listing 6-22:

```
Principals
=====
    Robbie Smith, Salary: $48,000.00, Bonus: $3,500.00
Instructors
=====
    Joan Carlson, Salary: $39,000.00
```

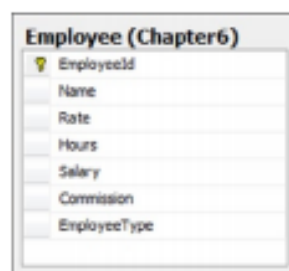
6-8. Model Nested Table per Hierarchy Inheritance

Masalah

Anda ingin memodelkan tabel menggunakan lebih dari satu tingkat Tabel per Hirarki Inheritance.

Solusi

Misalkan kita memiliki tabel Karyawan yang berisi berbagai jenis karyawan seperti Karyawan Per Jam dan Gaji, seperti yang ditunjukkan pada Gambar 6-10.



Gambar 6-10. Tabel Karyawan yang berisi berbagai jenis karyawan

Tabel Karyawan berisi karyawan per jam, karyawan yang digaji, dan karyawan yang ditugaskan, yang merupakan subtype karyawan yang digaji. Untuk memodelkan tabel ini dengan jenis turunan untuk karyawan yang dibayar per jam dan gaji serta jenis karyawan yang ditugaskan dari karyawan yang digaji, lakukan hal berikut:

1. Buat kelas baru di proyek Anda yang mewarisi dari DbContext.
2. Buat kelas entitas POCO untuk Employee, HourlyEmployee, SalariedEmployee, dan CommissionedEmployee, seperti yang ditunjukkan pada Listing 6-23.

Listing 6-23. Membuat Employee, HourlyEmployee, SalariedEmployee, dan CommissionedEmployee Entitas POCO

```
public abstract class Employee
{
    public int EmployeeId { get; set; }
    public string Name { get; set; }
}

public class SalariedEmployee : Employee
{
    public decimal? Salary { get; set; }
```

```

    }
    public class CommissionedEmployee : SalariedEmployee
    {
        public decimal? Commission { get; set; }
    }
    public class HourlyEmployee : Employee
    {
        public decimal? Rate { get; set; }
        public decimal? Hours { get; set; }
    }

```

3. Tambahkan properti otomatis tipe DbSet <Employee> ke subkelas DbContext Anda.
4. Ganti metode OnModelCreating dari DbContext untuk mengkonfigurasi nilai diskriminator TPH untuk setiap jenis turunan, seperti yang ditunjukkan pada Listing 6-24.

Listing 6-24. Overriding OnModelCreating Subclass DbContext untuk Mengkonfigurasi Nilai TPH Discriminator

```

protected override void OnModelCreating(DbModelBuilder
modelBuilder)
{
    base.OnModelCreating(modelBuilder);
    modelBuilder.Entity<Employee>()
        .HasKey(e => e.EmployeeId)
        .Property(e => e.EmployeeId)
        .HasDatabaseGeneratedOption(DatabaseGeneratedOption.Identity);
    modelBuilder.Entity<Employee>()
        .Map<HourlyEmployee>(m =>
            m.Requires("EmployeeType").HasValue("hourly"))
        .Map<SalariedEmployee>(m =>
            m.Requires("EmployeeType").HasValue("salaried"))
        .Map<CommissionedEmployee>(m =>
            m.Requires("EmployeeType").HasValue("commissioned"))
        .ToTable("Employee", "Chapter6");
}

```

Bagaimana itu bekerja

Tabel per Hirarki Inheritance adalah teknik pemodelan fleksibel. Kedalaman dan luasnya pohon inheritance dapat cukup besar dan mudah dilaksanakan. Pendekatan ini efisien karena tidak ada tabel tambahan dan keterlibatan yang diperlukan mereka dilibatkan.

Menerapkan TPH dengan pendekatan Code-First sangat mudah karena pewarisan yang berorientasi objek bersifat hierarkis.

Listing 6-25 menunjukkan memasukkan dan mengambil dari model kami.

Listing 6-25. Memasukkan dan Mengambil Entitas yang Berasal dari Karyawan

```
using (var context = new EF6RecipesContext())
{
    var hourly = new HourlyEmployee { Name = "Will Smith", Hours = 39,
    Rate = 7.75M };
    var salaried = new SalariedEmployee { Name = "JoAnn Woodland",
    Salary = 65400M };
    var commissioned = new CommissionedEmployee { Name = "Joel Clark",
    Salary = 32500M, Commission = 20M };
    context.Employees.Add(hourly);
    context.Employees.Add(salaried);
    context.Employees.Add(commissioned);
    context.SaveChanges();
}
using (var context = new EF6RecipesContext())
{
    Console.WriteLine("All Employees");
    Console.WriteLine("=====");
    foreach (var emp in context.Employees)
    {
        if (emp is HourlyEmployee)
            Console.WriteLine("{0} Hours = {1}, Rate = {2}/hour",
            emp.Name,
            ((HourlyEmployee)emp).Hours.Value.ToString(),
            ((HourlyEmployee)emp).Rate.Value.ToString("C"));
        else if (emp is CommissionedEmployee)
```

```

Console.WriteLine("{0} Salary = {1}, Commission = {2}%",
emp.Name,
((CommissionedEmployee)emp).Salary.Value.ToString("C"),
((CommissionedEmployee)emp).Commission.ToString());
else if (emp is SalariedEmployee)
Console.WriteLine("{0} Salary = {1}", emp.Name,
((SalariedEmployee)emp).Salary.Value.ToString("C"));
}
}

```

Output dari kode pada Listing 6-25 adalah sebagai berikut:

```

All Employees
*****
Will Smith Hours = 39.00, Rate = $7.75/hour
JoAnn Woodland Salary = $65,400.00
Joel Clark Salary = $32,500.00, Commission = 20.00%

```

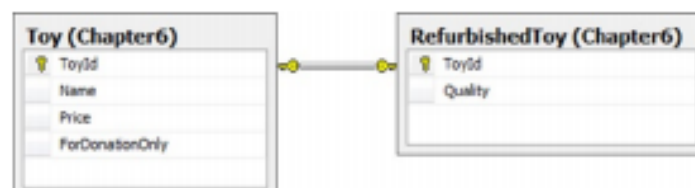
6-9. Menerapkan Ketentuan dalam Tabel per Tipe Inheritance

Masalah

Anda ingin menerapkan ketentuan saat menggunakan Tabel per Tipe Inheritance.

Solusi

Katakanlah Anda memiliki dua tabel yang digambarkan pada Gambar 6-11. Tabel Toy menggambarkan mainan yang diproduksi oleh perusahaan. Kebanyakan mainan yang diproduksi oleh perusahaan dijual. Beberapa mainan dibuat hanya untuk disumbangkan ke badan amal yang layak. Selama proses pembuatan, mainan mungkin rusak. Rusak mainan diperbaharui, dan inspektur menentukan kualitas yang dihasilkan dari mainan yang diperbaharui.

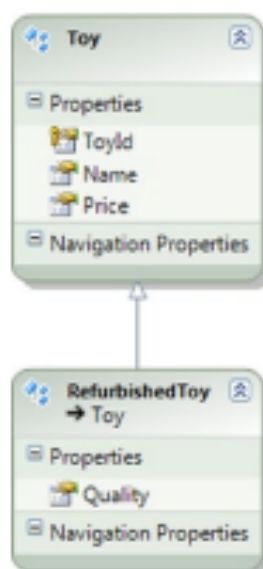


Gambar 6-11. Meja Toy dan RefurbishedToy dengan hubungan one-to-one

Aplikasi yang menghasilkan laporan untuk perusahaan tidak perlu mengakses mainan yang diproduksi untuk sumbangan. Untuk membuat model yang memfilter mainan yang diproduksi untuk donasi saat mewakili tabel Toy dan RefurbishedToy menggunakan Table per Type inheritance, lakukan hal berikut:

1. Tambahkan Model Data Entitas ADO.NET baru ke proyek Anda, dan impor tabel Urutan dan Pencarian.
2. Hapus hubungan antara Toy dan RefurbishedToy.
3. Klik kanan entitas Toy, dan pilih Add ► Inheritance. Pilih Toy sebagai entitas dasar dan RefurbishedToy sebagai entitas turunan.
4. Hapus properti ToyId di entitas RefurbishedToy.
5. Pilih entitas RefurbishedToy. Di jendela Detail Pemetaan, petakan kolom ToyId ke properti ToyId. Nilai ini akan berasal dari entitas basis Toy.
6. Hapus properti skalar ForDonationOnly dari entitas Toy.
7. Pilih entitas Toy, dan lihat jendela Detail Pemetaan. Gunakan Tambahkan Tabel atau Tampilan untuk memetakan entitas ini ke tabel Toy. Tambahkan kondisi Saat ForDonationOnly = 0.

Model yang dihasilkan ditunjukkan pada Gambar 6-12.



Gambar 6-12. Model selesai dengan entitas Toy dan entitas RefurbishedToy berasal

Bagaimana itu bekerja

Kami membatasi instance `RefurbishedToy` ke `toy` nondonation dengan menerapkan ketentuan pada entitas dasar. Pendekatan ini berguna dalam kasus seperti ketika kita perlu menerapkan filter permanen ke struktur inheritance sambil menggunakan tabel terpisah untuk menerapkan beberapa tipe turunan.

Kode dalam Listing 6-26 menunjukkan memasukkan dan mengambil dari model kami.

Listing 6-26. Memasukkan ke dalam dan Mengambil dari Model Kami

```
using (var context = new EF6RecipesContext())
{
    Context.Database.ExecuteSqlCommand(@"insert into chapter6.toy
    (Name,ForDonationOnly) values ('RagDoll',1)");
    var toy = new Toy { Name = "Fuzzy Bear", Price = 9.97M };
    var refurb = new RefurbishedToy { Name = "Derby Car", Price =
    19.99M,
    Quality = "Ok to sell" };
    context.Toys.Add(toy);
    context.Toys.Add(refurb);
    context.SaveChanges();
}
using (var context = new EF6RecipesContext())
{
    Console.WriteLine("All Toys");
    Console.WriteLine("=====");
    foreach (var toy in context.Toys)
    {
        Console.WriteLine("{0}", toy.Name);
    }
    Console.WriteLine("\nRefurbished Toys");
    foreach (var toy in context.Toys.OfType<RefurbishedToy>())
    {
        Console.WriteLine("{0}, Price = {1}, Quality = {2}", toy.Name,
        toy.Price, ((RefurbishedToy)toy).Quality);
    }
}
```

Berikut ini adalah output dari Listing 6-26:

```
All Toys
*****
Fuzzy Bear
Derby Car

Refurbished Toys
Derby Car, Price = 19.99, Quality = Ok to sell
```

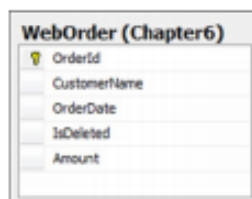
6-10. Membuat Filter pada Beberapa Kriteria

Masalah

Anda ingin memfilter baris untuk entitas berdasarkan beberapa kriteria.

Solusi

Anggaplah kita memiliki tabel yang memegang pesanan web, seperti yang ditunjukkan pada Gambar 6-13.



| |
|--------------|
| OrderId |
| CustomerName |
| OrderDate |
| IsDeleted |
| Amount |

Gambar 6-13. Tabel WebOrder yang berisi informasi tentang pesanan web

Misalkan kita memiliki persyaratan bisnis, yang mendefinisikan contoh WebOrder sebagai pesanan yang ditempatkan setelah hari pertama 2012 atau pesanan yang ditempatkan antara tahun 2010 dan 2012 yang tidak dihapus atau pesanan yang ditempatkan sebelum tahun 2010 yang memiliki jumlah pesanan lebih besar dari \$ 200. Filter semacam ini tidak dapat dibuat menggunakan kondisi yang agak terbatas yang tersedia di jendela Detail Pemetaan di perancang. Salah satu cara untuk menerapkan filter kompleks ini adalah dengan menggunakan QueryView. Untuk memodelkan entitas ini dan menerapkan filter yang memenuhi persyaratan bisnis menggunakan QueryView, lakukan hal berikut:

1. Tambahkan Model Data Entitas ADO.NET baru ke proyek Anda, dan impor tabel WebOrder. Buat prosedur tersimpan dalam Listing 6-27. Dalam dua langkah berikutnya, kami akan memetakan ini ke action insert, update, dan delete untuk entitas WebOrder.

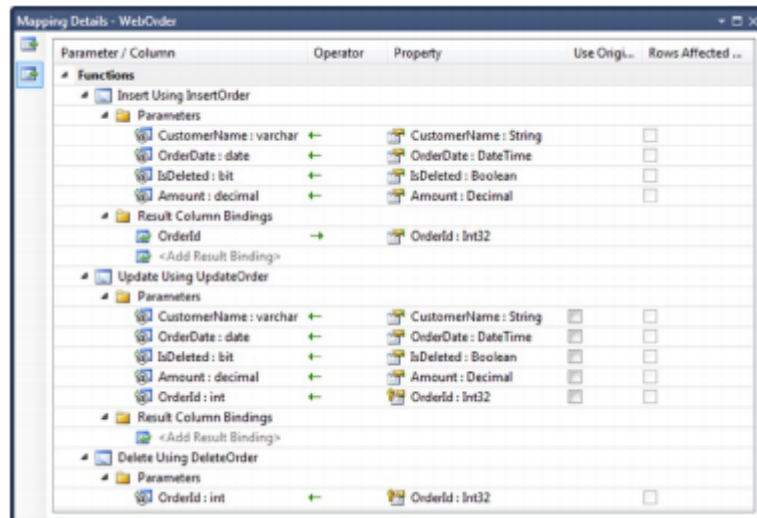
Listing 6-27. Prosedur Ditetapkan dalam Database untuk action Insert, Update, dan Delete di Entitas WebOrder

```
create procedure [Chapter6].[InsertOrder]
(@CustomerName varchar(50),@OrderDate date,@IsDeleted bit,@Amount
decimal)
as
begin
insert into chapter6.WebOrder (CustomerName, OrderDate,
IsDeleted, Amount)
values (@CustomerName, @OrderDate, @IsDeleted, @Amount)
select SCOPE_IDENTITY() as OrderId
end
go
create procedure [Chapter6].[UpdateOrder]
(@CustomerName varchar(50),@OrderDate date,@IsDeleted bit,
@Amount decimal, @OrderId int)
as
begin
update chapter6.WebOrder set CustomerName = @CustomerName,
OrderDate = @OrderDate,IsDeleted = @IsDeleted,Amount = @Amount
where OrderId = @OrderId
end
go
create procedure [Chapter6].[DeleteOrder]
(@OrderId int)
as
begin
delete from Chapter6.WebOrder where OrderId = @OrderId
end
```

2. Klik kanan permukaan desain, dan pilih Perbarui Model dari Database. Di Update Wizard, pilih prosedur yang tersimpan dalam InsertOrder, UpdateOrder, dan DeleteOrder.
3. Pilih entitas WebOrder, dan pilih tombol Map To Functions di jendela Detail Pemetaan. Tombol ini adalah yang kedua dari dua tombol di sisi kiri jendela. Petakan prosedur InsertOrder ke action Insert, prosedur UpdateOrder ke action Update, dan prosedur DeleteOrder ke action Delete. Pemetaan properti / parameter harus berbaris secara otomatis. Namun, nilai kembalian dari prosedur InsertOrder harus dipetakan ke properti

OrderId. Ini digunakan oleh Entity Framework untuk mendapatkan nilai kolom identitas OrderId setelah insert.

Gambar 6-14 menunjukkan pemetaan yang benar.



Gambar 6-14. Detail untuk pemetaan prosedur / tindakan yang tersimpan

4. Pilih pemetaan tabel (tombol atas) di jendela Detail Pemetaan. Hapus pemetaan ke tabel WebOrder. Kami akan memetakan ini menggunakan QueryView.

Klik kanan file .edmx di jendela Solution Explorer, dan pilih Open With ► Editor XML. Pada layer pemetaan C-S, di dalam tag <EntitySetMapping>, masukkan kode yang ditunjukkan pada Listing 6-28. Ini adalah QueryView yang akan memetakan entitas WebOrder kami.

Hati-hati! Perubahan yang dibuat ke lapisan pemetaan C-S akan hilang jika Anda melakukan Model Pembaruan lain dari Database.

Listing 6-28. Entity Set Mapping Menggunakan QueryView untuk WebOrder Table

```
<EntitySetMapping Name="WebOrders">
  <QueryView>
    select value
    EFRecipesModel.WebOrder(o.OrderId,
    o.CustomerName,o.OrderDate,o.IsDeleted,o.Amount)
    from EFRecipesModelStoreContainer.WebOrder as o
    where (o.OrderDate > datetime'2007-01-01 00:00') ||
    (o.OrderDate between cast('2005-01-01' as Edm.DateTime) and
    cast('2007-01-01' as Edm.DateTime) and !o.IsDeleted) ||
    (o.Amount > 800 and o.OrderDate <
```

```
cast('2005-01-01' as Edm.DateTime))  
</QueryView>  
</EntitySetMapping>
```

Bagaimana itu bekerja

QueryView adalah pemetaan read-only yang dapat digunakan sebagai pengganti pemetaan default yang ditawarkan oleh Entity Framework. Ketika QueryView berada di dalam tag <EntitySetMapping> dari layer pemetaan, ia memetakan entitas yang didefinisikan pada model store untuk entitas yang didefinisikan pada model konseptual. Ketika QueryView berada di dalam tag <AssociationSetMapping>, QueryView memetakan asosiasi yang didefinisikan pada model store ke asosiasi yang didefinisikan pada model konseptual. Satu penggunaan umum QueryView di dalam tag <AssociationSetMapping> adalah untuk menerapkan inheritance berdasarkan kondisi yang tidak didukung oleh pemetaan kondisi default.

QueryView dinyatakan dalam Entity SQL. QueryView dapat query hanya entitas yang didefinisikan pada model store. Selain itu, eSQL di QueryView tidak mendukung group by dan group aggregates.

Ketika entitas dipetakan menggunakan QueryView, Entity Framework tidak menyadari implementasi yang tepat dari pemetaan. Karena Entity Framework tidak tahu kolom dan tabel yang mendasari yang digunakan untuk membuat instance entitas, itu tidak dapat menghasilkan action store-level yang tepat untuk insert, update, atau delete entitas. Entity Framework melacak perubahan pada entitas ini setelah terwujud, tetapi tidak tahu cara mengubahnya di penyimpanan data yang mendasarinya.

Beban mengimplementasikan action insert, update, dan delete jatuh ke developer. Action ini bisa dilakukan diimplementasikan secara langsung dalam file .edmx atau mereka dapat diimplementasikan sebagai prosedur tersimpan dalam database yang mendasari. Untuk menghubungkan prosedur dengan action, Anda perlu membuat bagian <ModificationFunctionMapping>. Kami melakukan ini pada langkah 4 menggunakan desainer daripada langsung mengedit file .edmx.

Jika entitas yang dipetakan menggunakan QueryView memiliki asosiasi dengan entitas lain, asosiasi tersebut, bersama dengan entitas terkait, juga perlu dipetakan menggunakan QueryView. Tentu saja, ini bisa menjadi agak membosankan. QueryView adalah alat yang kuat, tetapi dapat dengan cepat menjadi memberatkan.

Beberapa kasus penggunaan umum untuk menggunakan QueryView adalah sebagai berikut.

1. Untuk menentukan filter yang tidak didukung secara langsung, seperti lebih besar dari, kurang dari, dan sebagainya
2. Untuk memetakan inheritance yang didasarkan pada kondisi selain null, bukan null, atau sama dengan
3. Untuk memetakan kolom yang dikomputasi atau mengembalikan subset kolom dari tabel, atau untuk mengubah pembatasan atau tipe data dari kolom, seperti menjadikannya nullable, atau untuk memunculkan kolom string sebagai integer
4. Untuk memetakan Table per Type Inheritance berdasarkan primary key dan foreign key yang berbeda
5. Untuk memetakan kolom yang sama dalam model penyimpanan ke beberapa jenis dalam model konseptual
6. Untuk memetakan beberapa jenis ke tabel yang sama

Di dalam QueryView di Listing 6-28, kami memiliki pernyataan SQL Entitas yang berisi tiga bagian. Bagian pertama adalah klausa pilih yang instantiate sebuah instance dari entitas WebOrder dengan konstruktor. Konstruktor mengambil nilai properti dalam urutan yang persis sama seperti yang didefinisikan pada model konseptual dalam Listing 6-29.

Listing 6-29. Definisi Entitas WebOrder dalam Model Konseptual

```
<EntityType Name="WebOrder">
  <Key>
    <PropertyRef Name="OrderId" />
  </Key>
  <Property Name="OrderId" Type="Int32" Nullable="false"
    annotation:StoreGeneratedPattern="Identity" />
  <Property Name="CustomerName" Type="String" Nullable="false"
    MaxLength="50" Unicode="false" FixedLength="false" />
```

```

<Property Name="OrderDate" Type="DateTime" Nullable="false" />
<Property Name="IsDeleted" Type="Boolean" Nullable="false" />
<Property Name="Amount" Type="Decimal" Nullable="false"
Precision="18" Scale="2" />
</EntityType>

```

Perhatikan bahwa, dalam Entity SQL di Listing 6-29 kami sepenuhnya memenuhi syarat namespace `EFRecipesModel` saat membuat instance dari entitas `WebOrder`. Namun, dari klausa kami juga memenuhi syarat store container, `EFRecipesModelStoreContainer`.

Bagian terakhir dari ekspresi SQL Entitas termasuk di mana klausa itu, tentu saja adalah seluruh alasan untuk menggunakan `QueryView` dalam contoh ini. Meskipun di mana klausa dapat arbitrarily complex, itu tunduk pada pembatasan untuk Entity SQL di `QueryView` seperti disebutkan di atas.

Kode dalam Listing 6-30 menunjukkan memasukkan dan mengambil `WebOrders`in model kami.

Listing 6-30. Memasukkan dan Mengambil Entitas `WebOrder`

```

using (var context = new EF6RecipesContext())
{
    var order = new WebOrder { CustomerName = "Jim Allen",
        OrderDate = DateTime.Parse("5/3/2012"),
        IsDeleted = false, Amount = 200 };
    context.WebOrders.Add(order);

    order = new WebOrder { CustomerName = "John Stevens",
        OrderDate = DateTime.Parse("1/1/2011"),
        IsDeleted = false, Amount = 400 };
    context.WebOrders.Add(order);

    order = new WebOrder { CustomerName = "Russel Smith",
        OrderDate = DateTime.Parse("1/3/2011"),
        IsDeleted = true, Amount = 500 };
    context.WebOrders.Add(order);

    order = new WebOrder { CustomerName = "Mike Hammer",
        OrderDate = DateTime.Parse("6/3/2013"),
        IsDeleted = true, Amount = 1800 };
    context.WebOrders.Add(order);
}

```

```

order = new WebOrder { CustomerName = "Steve Jones",
OrderDate = DateTime.Parse("1/1/2008"),
IsDeleted = true, Amount = 600 };
context.WebOrders.Add(order);
context.SaveChanges();
}
using (var context = new EF6RecipesContext())
{
Console.WriteLine("Orders");
Console.WriteLine("=====");
foreach (var order in context.WebOrders)
{
Console.WriteLine("\nCustomer: {0}", order.CustomerName);
Console.WriteLine("OrderDate: {0}",
order.OrderDate.ToShortDateString());
Console.WriteLine("Is Deleted: {0}", order.IsDeleted.ToString());
Console.WriteLine("Amount: {0}", order.Amount.ToString("C"));
}
}

```

Output dari kode dalam Listing 6-30 berikut. Perhatikan bahwa hanya pelanggan yang memenuhi kriteria yang kami tetapkan dalam ekspresi SQL Entitas di dalam QueryView ditampilkan.

```

Orders...

Customer: John Stevens
Order Date: 1/1/2011
Is Deleted: False
Amount: $400.00

Customer: Jim Allen
Order Date: 5/3/2012
Is Deleted: False
Amount: $200.00

Customer: Mike Hammer
Order Date: 6/3/2013
Is Deleted: True
Amount: $1,800.00

```

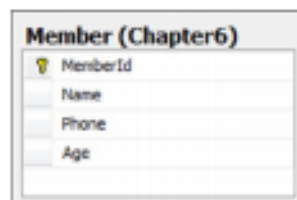
6-11. Menggunakan Kondisi Kompleks dengan Table per Hierarchy Inheritance

Masalah

Anda ingin memodelkan tabel menggunakan Tabel per hierarchy inheritance dengan menerapkan ketentuan yang lebih kompleks daripada yang didukung langsung oleh Entity Framework.

Solusi

Misalkan kita memiliki tabel Anggota, seperti yang digambarkan pada Gambar 6-15. Tabel Anggota menjelaskan anggota di klub kami. Dalam model kami, kami ingin mewakili anggota dewasa, anggota senior, dan anggota remaja sebagai jenis turunan menggunakan Table per Type inheritance.



Gambar 6-15. Tabel Anggota yang menjelaskan anggota di klub kami

Entity Framework mendukung Tabel per Hierarchy Inheritance berdasarkan kondisi =, null, dan tidak null. Ekspresi sederhana seperti <, antara, dan> tidak didukung. Dalam kasus kami, anggota yang usianya kurang dari 20 adalah remaja (usia minimum di klub kami adalah 13). Seorang anggota antara usia 20 dan 55 adalah orang dewasa. Dan, seperti yang Anda duga, seorang anggota yang berusia di atas 55 tahun adalah seorang senior. Untuk membuat model untuk tabel anggota dan tiga tipe turunan, lakukan hal berikut:

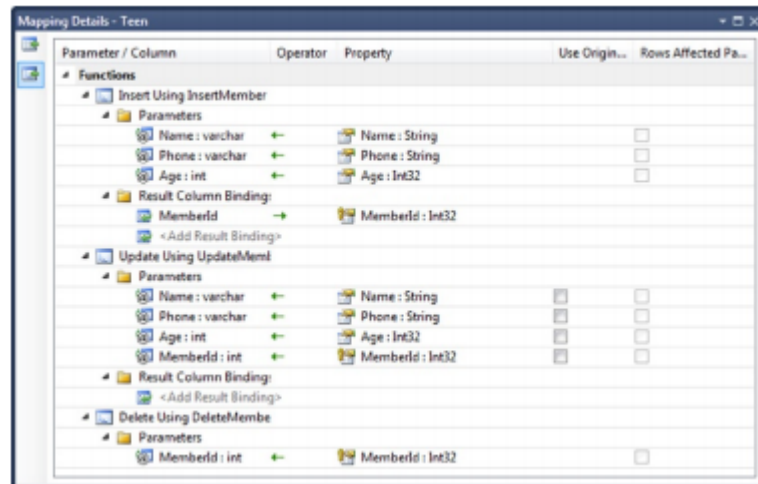
1. Tambahkan Model Data Entitas ADO.NET baru ke proyek Anda, dan impor tabel Anggota.
2. Klik kanan entitas Anggota, dan pilih Properties. Set atribut Abstrak ke true. Ini menandai entitas Anggota sebagai abstrak.
3. Buat prosedur tersimpan dalam Listing 6-31. Kami akan menggunakannya untuk menangani tindakan Insert, Update, dan Delete pada entitas yang kami peroleh dari entitas Anggota.

Listing 6-31. Stored Procedures untuk Insert, Update, dan Delete action

```
create procedure [chapter6].[InsertMember]
(@Name varchar(50), @Phone varchar(50), @Age int)
as
begin
insert into Chapter6.Member (Name, Phone, Age)
values (@Name,@Phone,@Age)
select SCOPE_IDENTITY() as MemberId
end
go
create procedure [chapter6].[UpdateMember]
(@Name varchar(50), @Phone varchar(50), @Age int, @MemberId int)
as
begin
update Chapter6.Member set Name=@Name, Phone=@Phone, Age=@Age
where MemberId = @MemberId
end
go
create procedure [chapter6].[DeleteMember]
(@MemberId int)
as
begin
delete from Chapter6.Member where MemberId = @MemberId
end
```

4. Klik kanan permukaan desain, dan pilih Update Model dari Database. Pilih prosedur tersimpan yang Anda buat pada langkah 3.
5. Klik kanan permukaan desain, dan pilih Add ► Entity. Beri nama entitas baru Teen, dan tetapkan tipe dasar ke Anggota. Ulangi langkah ini, buat entitas turunan Dewasa dan Senior.
6. Pilih entitas Anggota, dan lihat jendela Detail Pemetaan. Klik Maps to Member, dan pilih <Delete>. Ini menghapus pemetaan ke tabel Anggota.
7. Pilih entitas Teen, dan lihat jendela Detail Pemetaan. Klik tombol Map Entity to Functions. Ini adalah tombol bawah di sebelah kiri jendela Detail Pemetaan. Petakan prosedur tersimpan ke tindakan Insert, Update, dan Delete yang sesuai. Parameter / pemetaan properti akan secara otomatis terisi. Pastikan bahwa Anda mengatur Binding Kolom Hasil

untuk memetakan nilai kembalian ke properti MemberId untuk tindakan Insert. Kolom identitas ini dihasilkan di sisi database (lihat Gambar 6-16).



Gambar 6-16. Memetakan action Insert, Update, dan Delete untuk entitas Remaja

8. Ulangi langkah 7 untuk entitas Dewasa dan Senior.

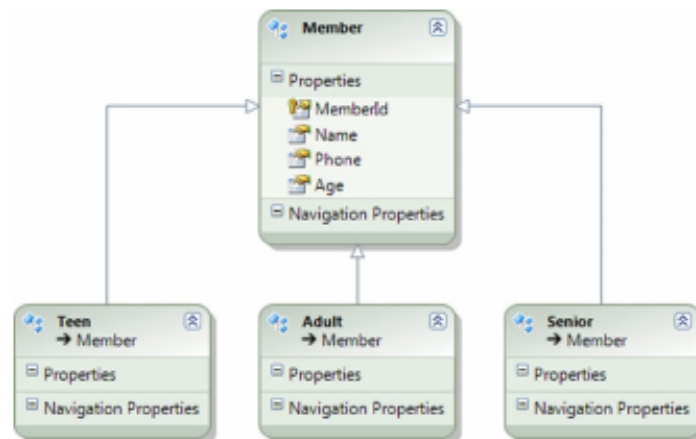
Klik kanan file .edmx di jendela Solution Explorer, dan pilih Open With ► XML Editor. Ini akan membuka file .edmx di XML editor.

9. Di bagian pemetaan C-S, di dalam tag <EntityContainerMapping>, masukkan kode QueryView yang ditunjukkan pada Listing 6-32.

Listing 6-32. QueryView untuk Memetakan Tabel Anggota ke Tipe Derived Teen, Adult, dan Senior

```
<EntitySetMapping Name="Members">
  <QueryView>
    select value
    case
    when m.Age < 20 then
      EFRecipesModel.Teen(m.MemberId,m.Name,m.Phone,m.Age)
    when m.Age between 20 and 55 then
      EFRecipesModel.Adult(m.MemberId,m.Name,m.Phone,m.Age)
    when m.Age > 55 then
      EFRecipesModel.Senior(m.MemberId,m.Name,m.Phone,m.Age)
    end
  from EFRecipesModelStoreContainer.Member as m
</QueryView>
</EntitySetMapping>
```


Model yang dihasilkan akan terlihat seperti pada Gambar 6-17.



Gambar 6-17. Model yang dihasilkan dengan Anggota dan tiga jenis turunan: Senior, Dewasa, dan Remaja

Bagaimana itu bekerja

Entity Framework hanya mendukung satu set kondisi terbatas ketika memodelkan Tabel per Hirarki Inheritance. Dalam resep ini, kami memperluas ketentuan menggunakan QueryView untuk menentukan pemetaan kami sendiri antara tabel Anggota yang mendasari dan jenis turunan: Senior, Dewasa, dan Remaja. Ini ditunjukkan pada Listing 6-32.

Sayangnya, QueryView memiliki harga. Karena kami telah mendefinisikan pemetaan sendiri, kami juga mengambil tanggung jawab untuk menerapkan tindakan Insert, Update, dan Delete untuk jenis turunan. Ini tidak terlalu sulit dalam kasus kami.

Dalam Listing 6-31, kami mendefinisikan prosedur untuk menangani tindakan Insert, Delete, dan Update. Kita hanya perlu membuat satu set karena tindakan ini menargetkan tabel Anggota yang mendasari. Dalam resep ini, kami menerapkannya sebagai prosedur tersimpan dalam database yang mendasarinya. Kita bisa mengimplementasikannya dalam file .edmx.

Dengan menggunakan perancang, kami memetakan prosedur ke tindakan Insert, Update, dan Delete untuk masing-masing jenis turunan. Ini melengkapi pekerjaan tambahan yang perlu kita lakukan ketika kita menggunakan QueryView.

Kode dalam Listing 6-33 menunjukkan memasukkan dan mengambil dari model kami. Di sini kita memasukkan satu contoh dari masing-masing jenis turunan kita. Di sisi pengambilan, kami mencetak anggota bersama dengan nomor telepon mereka, kecuali anggotanya adalah Remaja.

Listing 6-33. Memasukkan ke dalam dan Mengambil dari Model Kami

```
using (var context = new EF6RecipesContext())
{
    var teen = new Teen { Name = "Steven Keller", Age = 17,
        Phone = "817 867-5309" };
    var adult = new Adult { Name = "Margret Jones", Age = 53,
        Phone = "913 294-6059" };
    var senior = new Senior { Name = "Roland Park", Age = 71,
        Phone = "816 353-4458" };
    context.Members.Add(teen);
    context.Members.Add(adult);
    context.Members.Add(senior);
    context.SaveChanges();
}

using (var context = new EF6RecipesContext())
{
    Console.WriteLine("Club Members");
    Console.WriteLine("=====");
    foreach (var member in context.Members)
    {
        bool printPhone = true;
        string str = string.Empty;
        if (member is Teen)
        {
            str = " a Teen";
            printPhone = false;
        }
        else if (member is Adult)
            str = "an Adult";
        else if (member is Senior)
            str = "a Senior";
```

```

Console.WriteLine("{0} is {1} member, phone: {2}", member.Name,
str, printPhone ? member.Phone : "unavailable");
}
}

```

Berikut ini adalah output dari kode pada Listing 6-33:

```

Members of our club
=====
Steven Keller is a Teen member, phone: unavailable
Margret Jones is an Adult member, phone: 913 294-6059
Roland Park is a Senior member, phone: 816 353-4458

```

Penting untuk dicatat di sini bahwa tidak ada waktu desain, atau bahkan pengecekan runtime, dilakukan untuk memverifikasi usia untuk tipe turunan. Sangat mungkin untuk membuat instance dari tipe Remaja dan mengatur properti usia menjadi 74 — jelas bukan remaja. Namun, di sisi pengambilan, baris ini akan terwujud sebagai anggota Senior — situasi yang cenderung menyinggung anggota Remaja kami.

Kami dapat memperkenalkan validasi sebelum perubahan dilakukan pada penyimpanan data. Untuk melakukan ini, daftar untuk SavingChanges Event ketika konteks dibuat. Kami mentransfer acara ini ke kode kami yang melakukan validasi. Kode ini ditunjukkan pada Listing 6-34.

Listing 6-34. Penanganan Validasi dalam SavingChanges Event

```

public partial class EF6RecipesContext
{
    partial void OnContextCreated()
    {
        this.SavingChanges += new EventHandler(Validate);
    }

    public void Validate(object sender, EventArgs e)
    {
        var entities = this.ObjectStateManager
            .GetObjectStateEntries(EntityState.Added |
            EntityState.Modified)
            .Select(et => et.Entity as Member);
        foreach (var member in entities) {
            if (member is Teen && member.Age > 19) {

```

```
throw new ApplicationException("Entity validation failed");
}
else if (member is Adult && (member.Age < 20 || member.Age >= 55)) {
throw new ApplicationException("Entity validation failed");
}
else if (member is Senior && member.Age < 55) {
throw new ApplicationException("Entity validation failed");
}
}
}
}
}
```

Dalam Listing 6-34, saat `SaveChanges()` dipanggil, metode `Validate()` kami memeriksa setiap entitas yang telah ditambahkan atau dimodifikasi. Untuk masing-masing ini, kami memverifikasi bahwa properti usia sesuai untuk jenis entitas. Ketika kami menemukan kesalahan validasi, kami hanya melempar pengecualian.

Kami memiliki beberapa resep di Bab 12 yang fokus pada penanganan peristiwa dan memvalidasi objek sebelum mereka berkomitmen ke database.

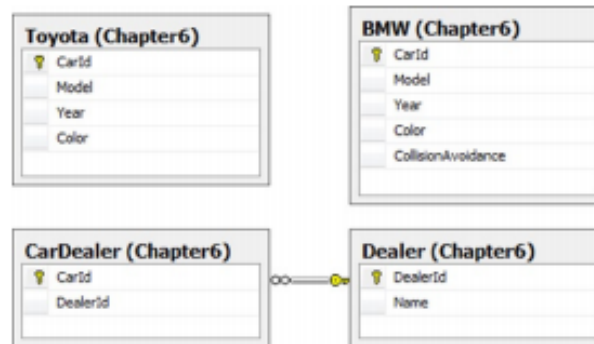
6-12. Tabel Pemodelan per Concrete Type Inheritance

Masalah

Anda memiliki dua atau lebih tabel dengan skema dan data yang serupa, dan Anda ingin memodelkan tabel ini sebagai jenis yang berasal dari entitas umum menggunakan Table per Concrete Type inheritance.

Solusi

Anggaplah kita memiliki tabel yang ditunjukkan pada Gambar 6-18.



Gambar 6-18. Tabel Toyota dan BMW dengan struktur serupa yang akan menjadi tipe turunan dari entitas Mobil

Pada Gambar 6-18, tabel Toyota dan BMW memiliki skema serupa dan merepresentasikan data serupa. Tabel BMW memiliki kolom tambahan dengan nilai bit yang menunjukkan apakah instance memiliki fitur tabrakan-penghindaran. Kami ingin membuat model dengan entitas dasar yang memegang sifat umum dari tabel Toyota dan BMW. Selain itu, kami ingin mewakili hubungan one-to-many antara dealer mobil dan mobil yang disimpan dalam inventaris. Gambar 6-22 menunjukkan model terakhir.

Untuk membuat model, lakukan hal berikut:

1. Tambahkan Model Data Entitas ADO.NET baru ke proyek Anda, dan impor tabel Toyota, BMW, CarDealer, dan Dealer.
2. Klik kanan permukaan desain, dan pilih Add ► Entity. Beri nama Kendaraan entitas baru, dan batalkan pilihan kotak centang Buat properti kunci.
3. Klik kanan entitas Mobil, dan lihat propertinya. Set properti Abstrak menjadi true.
4. Pindahkan sifat umum entitas Toyota dan BMW ke entitas Mobil. Anda dapat menggunakan Cut / Paste untuk memindahkan properti ini. Pastikan hanya CollisionAvoidance properti tetap dengan entitas BMW dan entitas Toyota tidak memiliki properti. Kedua entitas ini akan mewarisi sifat umum ini dari entitas Mobil.
5. Klik kanan entitas Mobil, dan pilih Add ► Inheritance. Atur entitas dasar sebagai Mobil dan entitas turunan sebagai BMW.
6. Ulangi langkah 5, tetapi kali ini tetapkan Toyota sebagai entitas turunan.
7. Klik kanan entitas CarDealer dan pilih Delete. Ketika diminta untuk menghapus tabel CarDealer dari store model, pilih Tidak.

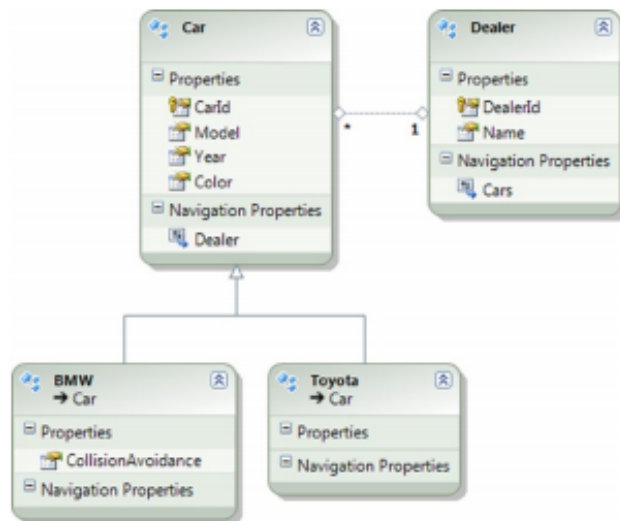
8. Klik kanan permukaan desain, dan pilih Add ► Association. Beri nama asosiasi CarDealer. Pilih Dealer di sebelah kiri dengan banyak sekali. Pilih Mobil di sebelah kanan dengan banyak ragam. Beri nama properti navigasi di Dealer sisi Mobil. Nama properti navigasi di sisi Mobil Dealer. Pastikan untuk menghapus centang pada Tambahkan properti foreign key.
9. Pilih asosiasi, dan lihat jendela Detail Pemetaan. Pilih CarDealer di menu drop-down Tambahkan Tabel atau Tampilan. Pastikan bahwa properti DealerId memetakan ke kolom DealerId, dan peta properti CarId ke kolom CarId.

Klik kanan file .edmx, dan pilih Open With ► Editor XML. Edit bagian pemetaan dengan perubahan yang ditampilkan dalam Listing 6-35 untuk entitas BMW dan Toyota.

Listing 6-35. Memetakan Tabel BMW dan Toyota

```
<EntitySetMapping Name="Cars">
  <EntityTypeMapping TypeName="IsTypeOf(EFRecipesModel.BMW)">
    <MappingFragment StoreEntitySet="BMW">
      <ScalarProperty Name="CollisionAvoidance"
        ColumnName="CollisionAvoidance" />
      <ScalarProperty Name="CarId" ColumnName="CarId"/>
      <ScalarProperty Name="Model" ColumnName="Model"/>
      <ScalarProperty Name="Year" ColumnName="Year"/>
      <ScalarProperty Name="Color" ColumnName="Color"/>
    </MappingFragment>
  </EntityTypeMapping>
  <EntityTypeMapping TypeName="IsTypeOf(EFRecipesModel.Toyota)">
    <MappingFragment StoreEntitySet="Toyota">
      <ScalarProperty Name="CarId" ColumnName="CarId"/>
      <ScalarProperty Name="Model" ColumnName="Model"/>
      <ScalarProperty Name="Year" ColumnName="Year"/>
      <ScalarProperty Name="Color" ColumnName="Color"/>
    </MappingFragment>
  </EntityTypeMapping>
</EntitySetMapping>
```

Model yang dihasilkan ditunjukkan pada Gambar 6-19.



Gambar 6-19. Model selesai dengan entitas yang diturunkan BMW dan Toyota diwakili dalam database sebagai tabel terpisah

Bagaimana itu bekerja

Tabel per Concrete Type adalah model pewarisan yang menarik karena memungkinkan setiap entitas turunan untuk memetakan ke tabel fisik terpisah. Dari perspektif praktis, tabel harus membagi setidaknya beberapa bagian dari skema umum. Skema umum ini dipetakan dalam entitas dasar sementara bagian skema tambahan dipetakan dalam entitas turunan. Untuk Table per Concrete Type inheritance agar berfungsi dengan benar, kunci entitas harus unik di seluruh tabel.

Entitas dasar ditandai abstrak, dan tidak dipetakan ke tabel mana pun. Dalam Tabel per concrete type, hanya entitas turunan yang dipetakan ke tabel.

Dalam contoh kami, kami menandai entitas Mobil sebagai abstrak dan kami tidak memetakannya ke tabel mana pun. Dalam pemetaan yang ditunjukkan pada Listing 6-35, perhatikan bahwa kami memetakan hanya entitas turunan BMW dan Toyota. Kami memindahkan semua properti umum (CarId, Model, Tahun, dan Warna) ke entitas dasar. Entitas yang diturunkan hanya berisi properti yang unik untuk entitas. Misalnya, entitas BMW memiliki properti CollisionAvoidance tambahan.

Karena entitas Toyota dan BMW berasal dari entitas Mobil, mereka menjadi bagian dari kumpulan entitas Mobil yang sama. Ini berarti bahwa kunci entitas CarId harus unik dalam kumpulan entitas yang sekarang berisi semua entitas turunan. Karena entitas dipetakan ke tabel yang berbeda, ada kemungkinan kita dapat memiliki tabrakan pada kunci. Untuk menghindarinya, kami menetapkan kolom CarId di setiap tabel sebagai kolom identitas. Untuk tabel BMW, kita atur seed awal ke 1 dengan kenaikan 2. Ini akan membuat nilai ganjil untuk kunci CarId. Untuk tabel Toyota, kami menetapkan seed awal menjadi 2 dengan tambahan 2. Ini akan membuat nilai event untuk kunci CarId.

Ketika memodelkan hubungan dalam Tabel per concrete type, lebih baik untuk mendefinisikannya pada tipe turunan daripada pada tipe dasar. Ini karena runtime Entity Framework tidak akan mengetahui tabel fisik mana yang mewakili ujung lain dari asosiasi. Dalam contoh kami, tentu saja, kami menyediakan tabel terpisah (CarDealer) yang berisi hubungan. Ini memungkinkan kami untuk memodelkan hubungan pada entitas dasar dengan memetakan asosiasi ke tabel CarDealer.

Ada banyak aplikasi praktis dari Table per Concrete Type inheritance Mungkin yang paling umum adalah bekerja dengan data arsip. Bayangkan Anda memiliki beberapa tahun pesanan untuk situs eCommerce Anda. Pada akhir setiap tahun, Anda mengarsipkan pesanan untuk 12 bulan sebelumnya dalam tabel arsip dan memulai Tahun Baru dengan tabel kosong. Dengan Table per Concrete Type inheritance, Anda dapat memodelkan perintah saat ini dan yang diarsipkan menggunakan pendekatan yang ditunjukkan di sini.

Table per Concrete Type inheritance memiliki keunggulan kinerja yang sangat penting dibanding inheritance lainnya model Ketika query jenis turunan, query yang dihasilkan menargetkan tabel yang mendasari spesifik tanpa bergabung tambahan Tabel per type inheritance atau penyaringan Tabel per Hierarki. Untuk dataset atau model besar dengan beberapa tipe turunan, keunggulan kinerja ini bisa signifikan.

Kerugian dari Table per Concrete Type inheritance termasuk overhead dari data yang berpotensi duplikat di seluruh tabel dan kompleksitas dalam mengasuransikan kunci unik di seluruh tabel. Dalam skenario arsip, data tidak diduplikasi tetapi hanya tersebar di beberapa tabel. Dalam skenario lain, data (properti) dapat diduplikasi di seluruh tabel.

Kode dalam Listing 6-36 menunjukkan memasukkan dan mengambil dari model kami.

Listing 6-36. Memasukkan dan Meminta Model Kami

```
using (var context = new EF6RecipesContext())
{
    var d1 = new Dealer { Name = "All Cities Toyota" };
    var d2 = new Dealer { Name = "Southtown Toyota" };
    var d3 = new Dealer { Name = "Luxury Auto World" };
    var c1 = new Toyota { Model = "Camry", Color = "Green",
        Year = "2014", Dealer = d1 };
    var c2 = new BMW { Model = "310i", Color = "Blue",
        CollisionAvoidance = true,
        Year = "2014", Dealer = d3 };
    var c3 = new Toyota { Model = "Tundra", Color = "Blue",
        Year = "2014", Dealer = d2 };
    context.Dealers.Add(d1);
    context.Dealers.Add(d2);
    context.Dealers.Add(d3);
    context.SaveChanges();
}

using (var context = new EF6RecipesContext())
{
    Console.WriteLine("Dealers and Their Cars");
    Console.WriteLine("=====");
    foreach (var dealer in context.Dealers)
    {
        Console.WriteLine("\nDealer: {0}", dealer.Name);
        foreach (var car in dealer.Cars)
        {
            string make = string.Empty;
            if (car is Toyota)
                make = "Toyota";
            else if (car is BMW)
                make = "BMW";
        }
    }
}
```

```

Console.WriteLine("\t{0} {1} {2} {3}", car.Year,
car.Color, make, car.Model);
}
}
}

```

Output dari kode dalam Listing 6-36 adalah sebagai berikut:

```

Dealers and Their Cars
=====

Dealer: Luxury Auto World
      2014 Blue BMW 310i

Dealer: Southtown Toyota
      2014 Blue Toyota Tundra

Dealer: All Cities Toyota
      2014 Green Toyota Camry

```

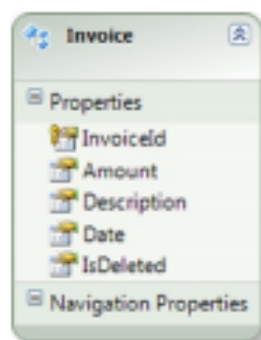
6-13. Menerapkan Ketentuan tentang Entitas Utama

Masalah

Anda ingin memperoleh entitas baru dari entitas dasar yang saat ini ada dalam model dan terus mengizinkan basis entitas yang akan dipakai.

Solusi

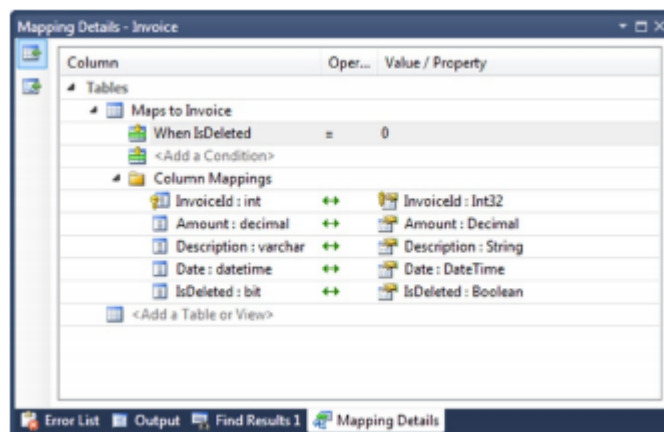
Anggaplah Anda memiliki model seperti yang ditunjukkan pada Gambar 6-20.



Gambar 6-20. Model kami dengan entitas Invoice

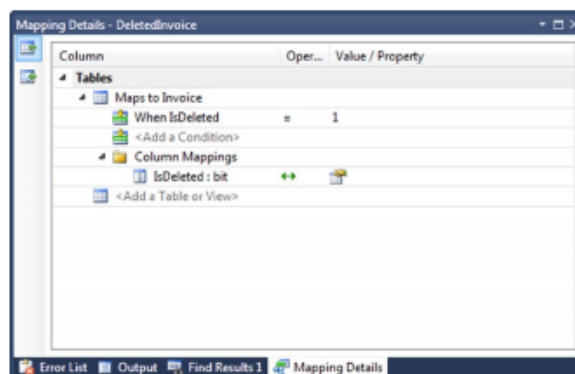
Model ini berisi entitas Invoice tunggal. Kami ingin memperoleh entitas baru yang mewakili faktur yang dihapus. Ini akan memungkinkan kita untuk memisahkan logika bisnis lebih bersih yang beroperasi pada faktur aktif berbeda dari pada faktur yang dihapus. Untuk menambahkan entitas turunan, lakukan hal berikut:

1. Lihat jendela Detail Pemetaan untuk entitas Faktur. Tambahkan kondisi pada kolom IsDeleted untuk memetakan entitas ketika kolom adalah 0, seperti yang ditunjukkan pada Gambar 6-21.



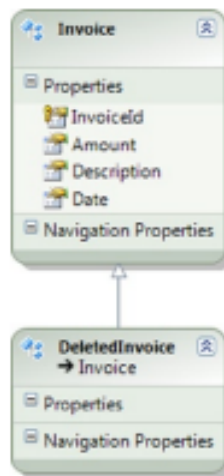
Gambar 6-21. Memetakan entitas Invoice ketika kolom IsDeleted adalah 0

2. Sekarang kolom IsDeleted digunakan dalam suatu kondisi, kita perlu menghapusnya dari properti skalar untuk entitas. Klik kanan properti IsDeleted dalam entitas dan pilih Hapus.
3. Klik kanan permukaan desain, dan pilih Add ➤ Entity. Beri nama entitas baru DeletedInvoice, dan pilih Faktur sebagai tipe dasar.
4. Lihat jendela Detail Pemetaan untuk entitas DeletedInvoice. Petakan entitas ke tabel Faktur. Tambahkan kondisi pada kolom IsDeleted untuk memetakan entitas ketika kolom adalah 1, seperti yang ditunjukkan pada Gambar 6-22.



Gambar 6-22. Memetakan entitas DeletedInvoice ke tabel Faktur ketika kolom IsDeleted adalah 1

Model terakhir dengan entitas Invoice dan entitas DeletedInvoice yang diturunkan ditunjukkan pada Gambar 6-23



Gambar 6-23. Model selesai kami dengan entitas Invoice dan entitas DeletedInvoice

Bagaimana itu bekerja

Ada dua cara berbeda untuk membuat model faktur kami dan menghapus faktur. Pendekatan yang kami perlihatkan di sini hanya disarankan jika Anda memiliki model dan basis kode yang sudah ada, dan Anda ingin menambahkan jenis turunan DeletedInvoice dengan dampak sekecil mungkin ke kode yang ada. Untuk model baru, akan lebih baik untuk mengambil tipe ActiveInvoice dan tipe DeletedInvoice dari tipe basis Invoice. Dalam pendekatan ini, Anda akan menandai tipe dasar sebagai abstrak.

Dengan menggunakan pendekatan yang kami tunjukkan di sini, Anda dapat dapat menentukan, seperti yang kami lakukan dalam kode pada Listing 6-37, jika entitas tersebut adalah DeletedInvoice, baik dengan casting atau dengan menggunakan metode OfType <> (). Namun, Anda tidak dapat memilih untuk entitas Faktur saja. Ini adalah kerugian kritis terhadap pendekatan yang kami tunjukkan di sini.

Pendekatan yang harus Anda gunakan untuk kode baru adalah menurunkan dua entitas baru: ActiveInvoice dan DeletelInvoice. Dengan dua tipe saudara ini, Anda dapat menggunakan metode casting atau OfType <> () untuk beroperasi pada kedua jenis secara seragam.

Listing 6-37. Menggunakan sebagai Operator untuk Menentukan Jika Kami Memiliki Invoice atau DeletedInvoice

```
using (var context = new EF6RecipesContext())
{
    context.Invoices.Add(new Invoice { Amount = 19.95M,
    Description = "Oil Change",
    Date = DateTime.Parse("4/11/13") });
    context.Invoices.Add(new Invoice { Amount = 129.95M,
    Description = "Wheel Alignment",
    Date = DateTime.Parse("4/01/13") });
    context.Invoices.Add(new DeletedInvoice { Amount = 39.95M,
    Description = "Engine Diagnosis",
    Date = DateTime.Parse("4/01/13") });
    context.SaveChanges();
}

using (var context = new EF6RecipesContext())
{
    foreach (var invoice in context.Invoices)
    {
        var isDeleted = invoice as DeletedInvoice;
        Console.WriteLine("{0} Invoice",
        isDeleted == null ? "Active" : "Deleted");
        Console.WriteLine("Description: {0}", invoice.Description);
        Console.WriteLine("Amount: {0}", invoice.Amount.ToString("C"));
        Console.WriteLine("Date: {0}", invoice.Date.ToShortDateString());
        Console.WriteLine();
    }
}
```

Berikut ini adalah hasil dari kode pada Listing 6-37:

```
Active Invoice
Description: Oil Change
Amount: $19.95
Date: 4/11/2013

Active Invoice
Description: Wheel Alignment
Amount: $129.95
Date: 4/1/2013

Deleted Invoice
Description: Engine Diagnosis
Amount: $39.95
Date: 4/1/2013
```

6-14. Menciptakan Asosiasi Independen dan Foreign Key

Masalah

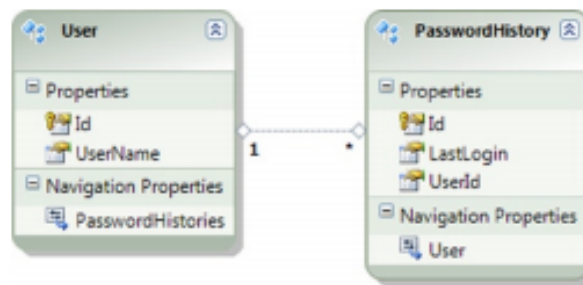
Anda ingin menggunakan Model Pertama untuk membuat asosiasi independen dan foreign key.

Solusi

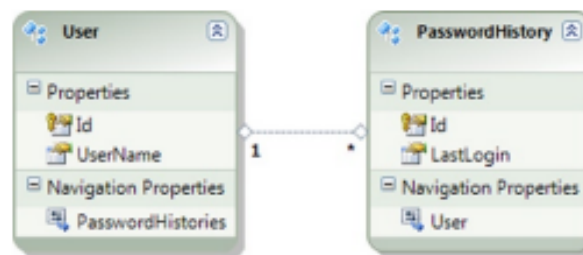
Foreign key dan asosiasi independen membantu kami menjaga integritas referensial dalam skema database dan menyediakan jalur navigasi ke entitas terkait. Untuk membuat foreign key dan asosiasi independen menggunakan Model Pertama, lakukan hal berikut:

1. Tambahkan Model Data Entitas ADO.NET baru ke proyek Anda. Pilih Empty Model ketika diminta untuk memilih isi model. Klik Selesai. Ini akan menciptakan permukaan desain yang kosong.
2. Klik kanan permukaan desain, dan pilih Add ► Entity. Nama Pengguna entitas baru dan klik OK.
3. Klik kanan entitas baru, dan tambahkan properti skalar untuk UserName.
4. Klik kanan permukaan desain, dan pilih Add ► Entity. Nama entitas Kata Kunci baru dan klik OK.
5. Klik kanan entitas baru, dan tambahkan properti skalar untuk LastLogin. Klik kanan properti LastLogin, dan ubah tipenya ke DateTime.
6. Klik kanan entitas Pengguna, dan pilih Add ► Asosiasi. Untuk membuat asosiasi foreign key, periksa properti Tambahkan foreign key ke kotak centang entitas Kata Sandi. Untuk membuat asosiasi independen, hapus centang pada kotak ini.
7. Klik kanan permukaan desain, dan pilih Generate Model from Database. Pilih koneksi database, dan selesaikan sisa wizard. Ini akan menghasilkan lapisan penyimpanan dan pemetaan model dan menghasilkan skrip untuk menghasilkan database untuk model.

Jika Anda memilih untuk membuat asosiasi foreign key, model akan terlihat seperti yang ditunjukkan pada Gambar 6-24. Jika Anda memilih untuk membuat asosiasi independen, model akan terlihat seperti yang ditunjukkan pada Gambar 6-25.



Gambar 6-24. Hubungan foreign key antara User dan PasswordHistory



Gambar 6-25. Asosiasi independen antara User dan PasswordHistory

Bagaimana itu bekerja

Dengan asosiasi foreign key, foreign key terekspos sebagai properti dalam entitas dependen. Membuka foreign key memungkinkan banyak aspek asosiasi dikelola dengan kode yang sama yang mengelola nilai properti lainnya. Ini sangat membantu dalam skenario terputus, seperti yang akan kita lihat di Bab 9. Asosiasi foreign key adalah default dalam Entity Framework.

Untuk asosiasi independen, foreign key tidak diekspos sebagai properti. Ini membuat pemodelan pada layer konseptual agak lebih bersih karena tidak ada noise yang diperkenalkan mengenai detail dari implementasi asosiasi. Dalam versi awal Entity Framework, hanya asosiasi independen yang didukung.

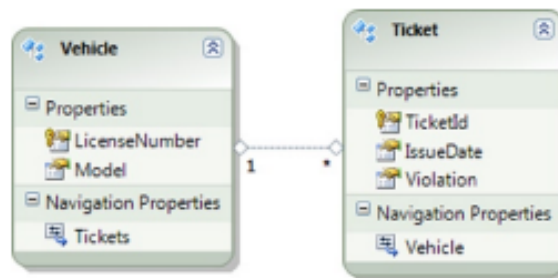
6-15. Mengubah Asosiasi Independen menjadi Asosiasi Foreign Key

Masalah

Anda memiliki model yang menggunakan asosiasi independen, dan Anda ingin mengubahnya menjadi asosiasi foreign key.

Solusi

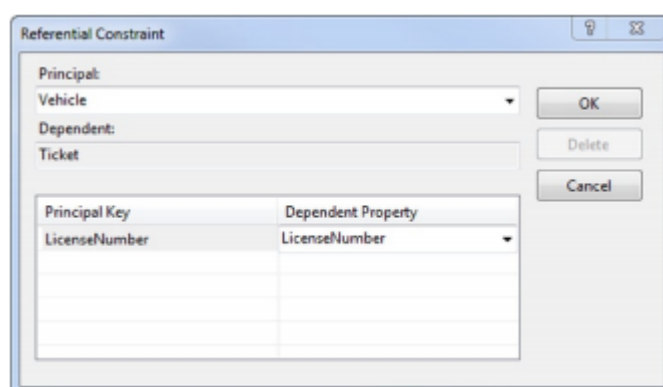
Katakanlah Anda memiliki model seperti yang ditunjukkan pada Gambar 6-26.



Gambar 6-26. Model untuk kendaraan dan tiket menggunakan asosiasi independen

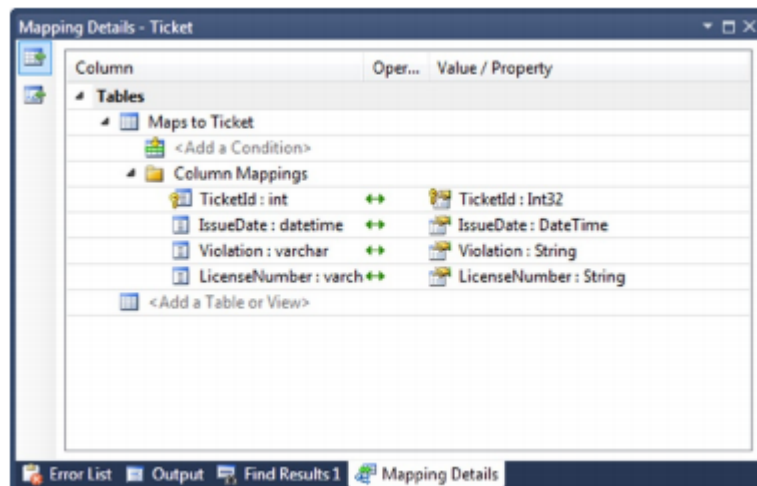
Untuk mengubah asosiasi dari asosiasi independen ke asosiasi foreign key, lakukan hal berikut:

1. Klik kanan entitas Tiket, dan pilih Add ► Scalar Property. Ganti nama property LicenseNumber.
2. Lihat jendela Detail Pemetaan untuk asosiasi. Hapus pemetaan ke tabel Tiket dengan memilih <Delete> dari kontrol Maps to Ticket.
3. Klik kanan asosiasi, dan lihat propertinya. Klik pada tombol Referential Constraint control. Di kotak dialog, pilih entitas Kendaraan di principal drop-down control. Kunci Utama dan Properti Tergantung keduanya harus ditetapkan ke LicenseNumber, seperti yang ditunjukkan pada Gambar 6-27.



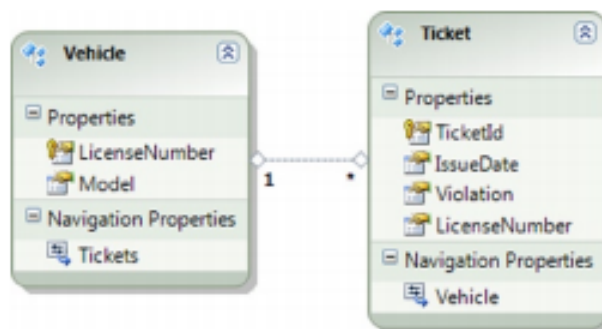
Gambar 6-27. Menciptakan batasan referensial untuk asosiasi foreign key

4. Lihat jendela Detail Pemetaan untuk entitas Tiket. Petakan kolom LicenseNumber ke properti LicenseNumber, seperti yang ditunjukkan pada Gambar 6-28.



Gambar 6-28. Memetakan kolom LicenseNumber ke properti LicenseNumber untuk entitas Tiket

Model terakhir ditunjukkan pada Gambar 6-29.



Gambar 6-29. Model dengan asosiasi independen berubah menjadi asosiasi foreign key

Bagaimana itu bekerja

Ketika Anda mengubah asosiasi independen menjadi asosiasi foreign key, sebagian besar kode yang ada akan terus berfungsi. Anda akan merasa lebih mudah sekarang untuk menghubungkan dua entitas hanya dengan mengatur foreign key yang terbuka ke nilai yang sesuai. Untuk mengubah hubungan dengan asosiasi independen, Anda perlu membuat instance EntityKey baru dan mengatur xxxReference.EntityKey entitas ke instance baru ini. Dengan asosiasi foreign key, Anda cukup mengatur properti foreign key yang terbuka ke nilai kunci.

Asosiasi foreign key saat ini tidak didukung untuk many-to-many asosiasi karena asosiasi ini harus dipetakan ke tabel tautan yang mendasarinya. Versi Entity Framework masa depan dapat mendukung asosiasi foreign key, bersama dengan payloads, untuk many-to-many asosiasi.