عنوان: طراحی روابط و ارجاعات در RavenDB
نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۱۸ ۱:۰
آدرس: www.dotnettips.info
برچسب‌ها: NoSQL, RavenDB

در قسمت‌های قبل، با پیش زمینه‌ی ذهنی طراحی مدل‌های RavenDB به همراه اصول مقدماتی کوئری نویسی آن آشنا شدیم. در این قسمت قصد داریم معادل‌های روابط موجود در بانک‌های اطلاعاتی رابطه‌ای را در RavenDB و مطابق ذهنیت غیر رابطه‌ای آن، مدلسازی کنیم و مثال‌های بیشتری را بررسی نمائیم.

مدیریت روابط در RavenDB

یکی از اصول طراحی مدل‌ها در RavenDB، مستقل بودن اسناد یا documents است. به این ترتیب کلیه اطلاعاتی که یک سند نیاز دارد، داخل همان سند ذخیره می‌شوند (به این نوع شیء، Root Aggregate هم گفته می‌شود). اما این اصل سبب نخواهد شد تا نتوان یا نباید ارتباطی را بین اسناد تعریف کرد. بنابراین سؤال مهم اینجا است که چه اطلاعات مرتبطی باید داخل یک سند ذخیره شوند و چه اطلاعاتی باید به سند دیگری ارجاع داده شوند. برای پاسخ به این سؤال سه روش ذیل را باید مدنظر داشت:

الف) Denormalized references

فرض کنید در دنیای رابطه‌ای دو جدول سفارش و مشتری را دارید. در این حالت، جدول سفارش تنها شماره آی دی اطلاعات مشتری را از جدول مشتری یا کاربران سیستم، در خود ذخیره خواهد کرد. به این ترتیب از تکرار اطلاعات مشتری در جدول سفارشات جلوگیری می‌گردد. اما اگر اطلاعات پرکاربرد مشتری را در داخل جدول سفارش قرار دهیم به آن denormalized reference گفته می‌شود.

ایجاد denormalized reference یکی از روش‌های مرسوم در دنیای NoSQL و RavenDB است؛ خصوصا جهت سهولت نمایش اطلاعات. به این ترتیب ارجاع به سندهای دیگر کمتر شده و ترافیک شبکه نیز کاهش می‌یابد. برای مثال در اینجا نام و آدرس مشتری را داخل سند ثبت شده قرار می‌دهیم و از سایر اطلاعات او (که اهمیت نمایشی ندارند) مانند کلمه عبور و امثال آن صرفنظر خواهیم کرد.

اینجا است که یک سری از سؤالات مطرح خواهند شد مانند : «اگر آدرس مشتری تغییر کرد، چطور؟»

بنابراین بهترین حالت استفاده از روش denormalized references محدود خواهد شد به موارد ذیل:

الف) قید اطلاعاتی که به ندرت تغییر می‌کنند. برای مثال نام یک شخص یا نام یک کشور، استان یا شهر.

ب) ثبت اطلاعات تکراری که در طول زمان تغییر می‌کنند، اما باید تاریخچه‌ی آن‌ها حفظ شوند. برای مثال اگر آدرس مشتری تغییر کرده است، واقعا اجناس سندهای قبلی او، صرفنظر از آدرس جدیدی که اعلام کرده است، به آدرس قبلی او ارسال شده‌اند و این تاریخچه باید در سیستم حفظ شوند.

ج) اطلاعاتی که ممکن است بعدها حذف شوند؛ اما نیاز است سابقه اسناد قبلی تخریب نشوند. برای مثال کارخانه‌ای را درنظر بگیرید که امسال یک سری چینی خاص را تولید می‌کند و می‌فروشد. سال بعد خط تولید خود را عوض کرده و سری اجناس دیگری را شروع به تولید و فروش خواهد کرد. در بانک‌های اطلاعاتی رابطه‌ای نمی‌توان اجناسی را که در جداول دیگر ارجاع دارند، به این سادگی‌ها حذف کرد. در اینجا باید از روش‌هایی مانند تعریف فیلد بیتی IsDeleted برای مخفی کردن ظاهری رکوردهای موجود کمک گرفت. اما در دنیای رابطه‌ای، اطلاعات مهم محصول را در سند اصلی ثبت کنید. بعد هر زمانیکه نیازی به محصول نبود، کلا تعریف آن‌را حذف نمائید.

ب) Includes

Includes در RavenDB برای پوشش مشکلات denormalization ارائه شده است. در اینجا بجای اینکه یک شیء کپی اطلاعات پرکاربرد شیءایی دیگر را در خود ذخیره کند، تنها ارجاعی (یک Id رشته‌ای) از آن شیء را در سند مرتبط ذخیره خواهد کرد.

```
public class Order
{
    public string CustomerId { get; set; }
    public LineItem[] LineItems { get; set; }
    public double TotalPrice { get; set; }
}

public class Customer
```

```
{
    public string Name { get; set; }
    public string Address { get; set; }
    public short Age { get; set; }
    public string HashedPassword { get; set; }
}
```

برای نمونه در کلاس Order شاهد یک Id رشته‌ای ارجاع دهنده به کلاس Customer هستیم. هرگاه که نیاز به بارگذاری اطلاعات شیء Order به همراه کل اطلاعات مشتری او تنها در یک رفت و برگشت به بانک اطلاعاتی باشد، می‌توان از متد الحاقی Include مختص RavenDB استفاده کرد:

```
var order = session.Include<Order>(x => x.CustomerId)
                   .Load("orders/1234");

این کوئری از کش سشن خوانده می‌شود و کاری به سرور ندارد //
var cust = session.Load<Customer>(order.CustomerId);
```

همانطور که مشاهده می‌کنید، با ذکر متد Include، اعلام کرده‌ایم که مایل هستیم تا اطلاعات سند مشتری متناظر را نیز داشته باشیم. در این حالت در Load بعدی که بر اساس Id مشتری انجام شده، دیگر رفت و برگشتی به سرور انجام نشده و اطلاعات مشتری از کش سشن جاری که پیشتر با فراخوانی Include مقدار دهی شده است، دریافت می‌گردد.
حتی می‌توان چند سند مرتبط را با هم بارگذاری کرد؛ با حداقل رفت و برگشت به سرور:

```
var orders = session.Include<Order>(x => x.CustomerId)
    .Load("orders/1234", "orders/4321");

foreach (var order in orders)
{
    // این کوئری‌ها سمت کلاینت هستند و به سرور ارسال نمی‌شوند
    var cust = session.Load<Customer>(order.CustomerId);
}
```

همچنین امکان استفاده از متد Include در LINQ API نیز پیش بینی شده است. برای این منظور باید از متد Customize استفاده کرد:

```
var orders = session.Query<Order>()
    .Customize(x => x.Include<Order>(o => o.CustomerId))
    .Where(x => x.TotalPrice > 100)
    .ToList();

foreach (var order in orders)
{
    // این کوئری‌ها سمت کلاینت اجرا می‌شوند
    var cust = session.Load<Customer>(order.CustomerId);
}
```

Includeهای یک به چند

اکنون فرض کنید به کلاس سفارش، آرایه تامین کننده‌ها نیز افزوده شده است (رابطه یک به چند):

```
public class Order
{
    public string CustomerId { get; set; }
    public string[] SupplierIds { get; set; }
    public LineItem[] LineItems { get; set; }
    public double TotalPrice { get; set; }
}
```

بارگذاری یکباره روابط یک به چند نیز با Include میسر است:

```
var orders = session.Include<Order>(x => x.SupplierIds)
    .Load("orders/1234", "orders/4321");

foreach (var order in orders)
{
    foreach (var supplierId in order.SupplierIds)
    {
        // از کش سشن خوانده می‌شود
        var supp = session.Load<Supplier>(supplierId);
    }
}
```

Includeهای چند سطحی

در اینجا کلاس سفارشی را در نظر بگیرید که دارای خاصیت ارجاع دهنده نیز هست. این خاصیت به شکل یک کلاس تعریف شده است و نه به شکل یک آی دی رشته‌ای:

```
public class Order
{
    public string CustomerId { get; set; }
    public string[] SupplierIds { get; set; }
    public Referral Refferal { get; set; }
    public LineItem[] LineItems { get; set; }
    public double TotalPrice { get; set; }
}

public class Referral
{
    public string CustomerId { get; set; }
    public double CommissionPercentage { get; set; }
}
```

متد Include امکان ارجاع به خواص تو در تو را نیز دارد:

```
var order = session.Include<Order>(x => x.Refferal.CustomerId)
    .Load("orders/1234");

// از کش سشن خوانده می‌شود
var referrer = session.Load<Customer>(order.Refferal.CustomerId);
```

همچنین این متد با مجموعه‌ها نیز کار می‌کند. برای مثال اگر تعریف متد LineItem به صورت زیر باشد:

```
public class LineItem
{
    public string ProductId { get; set; }
    public string Name { get; set; }
    public int Quantity { get; set; }
    public double Price { get; set; }
}
```

برای بارگذاری یکباره اسناد مرتبط می‌توان به روش ذیل عمل کرد:

```
var order = session.Include<Order>(x => x.LineItems.Select(li => li.ProductId))
    .Load("orders/1234");

foreach (var lineItem in order.LineItems)
{
    // از کش سمت کلاینت خوانده می‌شود
    var product = session.Load<Product>(lineItem.ProductId);
}
```

و به صورت خلاصه برای باگذاری اسناد مرتبط، دیگر از دو کوئری پشت سر هم ذیل استفاده نکنید:

```
var order = session.Load<Order>("orders/1");
var customer = session.Load<Customer>(order.CustomerId);
```

این دو کوئری یعنی دوبار رفت و برگشت به سرور. با استفاده از Include می‌توان تعداد رفت و برگشت‌ها و همچنین ترافیک شبکه را کاهش داد. به علاوه سرعت کار نیز افزایش خواهد یافت.

ج) تفاوت بین Reference و Relationship

برای درک اینکه آیا اطلاعات یک شیء مرتبط را بهتر است داخل شیء اصلی (Aggregate rooe) ذخیره کرد یا خیر، باید مفاهیم ارجاع و ارتباط را بررسی کنیم.
اگر به مثال سفارش و مشتری دقت کنیم، یک سفارش را بدون مشتری نیز می‌توان تکمیل کرد. برای مثال بسیاری از فروشگاه‌ها به همین نحو عمل می‌کنند و اگر شماره Id مشتری را به سندی اضافه می‌کنیم، صرفا جهت این است که بدانیم این سند متعلق به شخص دیگری نیست. بنابراین «ارجاعی» به کاربر در جدول سفارش می‌تواند وجود داشته باشد.
اکنون اقلام سفارش را درنظر بگیرید. هر آیتم سفارش تنها با بودن آن سفارش خاص است که معنا پیدا می‌کنند و نه بدون آن. این آیتم می‌تواند ارجاعی به محصول مرتبط داشته باشد. اینجا است که می‌گوییم اقلام سند با سفارش «در ارتباط» هستند؛ اما یک سند ارجاعی دارد به مشتری.
از این دو مفهوم برای تشخیص تشکیل Root Aggregate استفاده می‌شود. به این ترتیب تشخیص داده‌ایم اقلام سند، Root Aggregate را تشکیل می‌دهند؛ بنابراین ذخیره سازی تمام آن‌ها داخل یک سند RavenDB معنا پیدا می‌کند.

چند مثال برای درک بهتر نحوه طراحی اسناد در RavenDB

الف) Stackoverflow
صفحه نمایش یک سؤال و پاسخ‌های آن و همچنین رای‌های هر آیتم را درنظر بگیرید. در اینجا کاربران همزمانی ممکن است به یک سؤال رای بدهند، پاسخ‌هایی را ارائه دهند و یا کاربر اصلی، سؤال خویش را ویرایش کند. به این ترتیب با قرار دادن کلیه آیتم‌های این سند داخل آن، به مشکلات همزمانی برخواهیم خورد. برای مثال واقعا نمی‌خواهیم که به علت افزوده شدن یک پاسخ، کل سند قفل شود.
بنابراین ذخیره سازی سؤال در یک سند و ذخیره سازی لیست پاسخ‌ها در سندی دیگر، طراحی بهتری خواهد بود.

ب) سبد خرید و آیتم‌های آن
زمانیکه کاربری مشغول به خرید آنلاین از سایتی می‌شود، لیست اقلام انتخابی او یک سفارش را تشکیل داده و به تنهایی معنا پیدا نمی‌کنند. به همین جهت ذخیره سازی اقلام سفارش به صورت یک Root aggregate در اینجا مفهوم داشته و متداول است.

ج) یک بلاگ و کامنت‌های آن
در اینجا نیز کاربران، مجزای از مطلب اصلی ارسال شده ممکن است نظرات خود را ویرایش کنند یا اینکه بخواهیم نظرات را جداگانه لیست کنیم. بنابراین این دو (مطالب و نظرات) موضوعاتی جداگانه بوده و نیازی نیست به صورت یک Root aggregate تعریف شوند.

بنابراین در حین طراحی اسناد NoSQL باید به اعمال و «محدوده‌های تراکنشی» انجام شده دقت داشت تا اینکه صرفا عنوان شود این یک رابطه یک به چند یا چند به چند است.

نظرات خوانندگان

نویسنده: وحید نصیری
تاریخ: ۱۳۹۲/۰۶/۲۴ ۰:۴۳

یک مثال تکمیلی

شبیه سازی سایت Stackoverflow با RavenDB

توضیحات مفصل آن به صورت یک ویدیو