عنوان: **نکاتی در مورد استفاده از توابع تجمعی در Entity framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۰:۳۸
آدرس: www.dotnettips.info
برچسب‌ها: Entity framework

استفاده از Aggregate functions یا توابع تجمعی چه در زمان SQL نویسی مستقیم و یا در حالت استفاده از LINQ to Entities نیاز به ملاحظات خاصی دارد که عدم رعایت آن‌ها سبب کرش برنامه در زمان موعد خواهد شد. در ادامه تعدادی از این موارد را مرور خواهیم کرد.

ابتدا مدل‌های برنامه را در نظر بگیرید که از یک صورتحساب، به همراه ریز قیمت‌های آیتم‌های مرتبط با آن تشکیل شده است:

```
public class Bill
    {
        public int Id { set; get; }
        public string Name { set; get; }

        public virtual ICollection<Transaction> Transactions { set; get; }
    }

    public class Transaction
    {
        public int Id { set; get; }
        public DateTime AddDate { set; get; }
        public int Amount { set; get; }

        [ForeignKey("BillId")]
        public virtual Bill Bill { set; get; }
        public int BillId { set; get; }
    }
```

در ادامه این کلاس‌ها را در معرض دید EF Code first قرار می‌دهیم:

```
public class MyContext : DbContext
    {
        public DbSet<Bill> Bills { get; set; }
        public DbSet<Transaction> Transactions { get; set; }
    }
```

همچنین تعدادی رکورد اولیه را نیز جهت انجام آزمایشات به بانک اطلاعاتی متناظر، اضافه خواهیم کرد:

```
public class Configuration : DbMigrationsConfiguration<MyContext>
    {
        public Configuration()
        {
            AutomaticMigrationsEnabled = true;
            AutomaticMigrationDataLossAllowed = true;
        }

        protected override void Seed(MyContext context)
        {
            var bill1 = new Bill { Name = "bill-1" };
            context.Bills.Add(bill1);

            for (int i = 0; i < 11; i++)
            {
                context.Transactions.Add(new Transaction
                {
                    AddDate = DateTime.Now.AddDays(-i),
                    Amount = 1000000000 + i,
                    Bill = bill1
                });
            }
            base.Seed(context);
        }
    }
```

در اینجا به عمد از ارقام بزرگ استفاده شده است تا نمایانگر عملکرد یک سیستم واقعی در طول زمان باشد.

**اولین مثال: یک جمع ساده**

```
public static class Test
    {
        public static void RunTests()
        {
            Database.SetInitializer(new MigrateDatabaseToLatestVersion<MyContext, Configuration>());
            using (var context = new MyContext())
            {
                var sum = context.Transactions.Sum(x => x.Amount);
                Console.WriteLine(sum);
            }
        }
    }
```

ساده‌ترین نیازی را که در اینجا می‌توان مدنظر داشت، جمع کل تراکنش‌های سیستم است. به نظر شما خروجی کوئری فوق چیست؟

خروجی SQL کوئری فوق به نحو زیر است:

```
SELECT
        [GroupBy1].[A1] AS [C1]
        FROM ( SELECT
                  SUM([Extent1].[Amount]) AS [A1]
                  FROM [dbo].[Transactions] AS [Extent1]
                  )  AS [GroupBy1]
```

و خروجی واقعی آن استثنای زیر می‌باشد:

```
Arithmetic overflow error converting expression to data type int.
```

بله. محاسبه ممکن نیست؛ چون جمع حاصل از بازه اعداد صحیح خارج شده است.

**راه حل:**

نیاز است جمع را بر روی Int64 بجای Int32 انجام دهیم:

```
var sum2 = context.Transactions.Sum(x => (Int64)x.Amount);
```

```
SELECT
      [GroupBy1].[A1] AS [C1]
         FROM ( SELECT
                   SUM( CAST( [Extent1].[Amount] AS bigint)) AS [A1]
                   FROM [dbo].[Transactions] AS [Extent1]
               )  AS [GroupBy1]
```

**مثال دوم: سیستم باید بتواند با نبود رکوردها نیز صحیح کار کند**

برای نمونه کوئری زیر را بر روی بازه‌ای که سیستم عملکرد نداشته است، در نظر بگیرید:

```
var date = DateTime.Now.AddDays(10);
var sum3 = context.Transactions
                  .Where(x => x.AddDate > date)
                  .Sum(x => (Int64)x.Amount);
```

یک چنین خروجی SQL ایی دارد:

```
SELECT
     [GroupBy1].[A1] AS [C1]
        FROM ( SELECT
                    SUM( CAST( [Extent1].[Amount] AS bigint)) AS [A1]
                    FROM [dbo].[Transactions] AS [Extent1]
                    WHERE [Extent1].[AddDate] > @p__linq__0
              )  AS [GroupBy1]
```

اما در سمت کدهای ما با خطای زیر متوقف می‌شود:

```
The cast to value type 'Int64' failed because the materialized value is null.
Either the result type's generic parameter or the query must use a nullable type.
```

**راه حل:** استفاده از نوع‌های nullable در اینجا ضروری است:

```
var date = DateTime.Now.AddDays(10);
var sum3 = context.Transactions
                  .Where(x => x.AddDate > date)
                  .Sum(x => (Int64?)x.Amount) ?? 0;
```

به این ترتیب، خروجی صفر را بدون مشکل، دریافت خواهیم کرد.

**مثال سوم: حالت‌های خاص استفاده از خواص راهبری**

کوئری زیر را در نظر بگیرید:

```
var sum4 = context.Bills.First().Transactions.Sum(x => (Int64?)x.Amount) ?? 0;
```

در اینجا قصد داریم جمع تراکنش‌های صورتحساب اول را بدست بیاوریم که از طریق استفاده از خاصیت راهبری Transactions کلاس Bill، به نحو فوق میسر شده است. به نظر شما خروجی SQL آن به چه صورتی است؟

```
SELECT
     [Extent1].[Id] AS [Id],
     [Extent1].[AddDate] AS [AddDate],
     [Extent1].[Amount] AS [Amount],
     [Extent1].[BillId] AS [BillId]
   FROM [dbo].[Transactions] AS [Extent1]
   WHERE [Extent1].[BillId] = @EntityKeyValue1
```

بله! در اینجا خبری از Sum نیست. ابتدا کل اطلاعات دریافت شده و سپس جمع و منهای نهایی در سمت کلاینت بر روی آن‌ها انجام می‌شود؛ که بسیار ناکارآمد است. (قرار است این مورد ویژه، در نگارش‌های بعدی بهبود یابد)

**راه حل کنونی:**

```
var entry = context.Bills.First();
var sum5 = context.Entry(entry).Collection(x => x.Transactions).Query().Sum(x => (Int64?)x.Amount) ??
0;
```

در اینجا باید از روش خاصی که مشاهده می‌کنید جهت کار با خواص راهبری استفاده کرد و نکته اصلی آن استفاده از متد Query است. حاصل کوئری LINQ فوق اینبار SQL مطلوب زیر است که سمت سرور عملیات جمع را انجام می‌دهد و نه سمت کلاینت:

```
SELECT
    [GroupBy1].[A1] AS [C1]
     FROM ( SELECT
                SUM( CAST( [Extent1].[Amount] AS bigint)) AS [A1]
                    FROM [dbo].[Transactions] AS [Extent1]
                     WHERE [Extent1].[BillId] = @EntityKeyValue1
            )  AS [GroupBy1]
```

نکاتی که در اینجا ذکر شدند در مورد تمام توابع تجمعی مانند Sum، Count، Max و Min و غیره صادق هستند و باید به آن‌ها نیز دقت داشت.

## نظرات خوانندگان

نویسنده: رضا بزرگی
تاریخ: ۱۳۹۱/۰۷/۱۱ ۱۸:۵۴

ممنون از این سری پست‌ها. عالین.
و خیلی جالب نشون دادید که استفاده از پروفایلر چقدر اهمیت داره. مرسی.