عنوان: **کار با نوع داده‌ی HierarchyID توسط Entity framework**
نویسنده: وحید نصیری
تاریخ: ۱۳۹۳/۱۲/۱۹ ۱۶:۳۵
آدرس: www.dotnettips.info
گروه‌ها: Entity framework, SQL Server, Tree

نوع داده‌ی HierarchyID به همراه SQL Server 2008 برای کار با داده‌هایی با ساختار درختی ارائه شد. در حال حاضر هیچکدام از ORMهای موجود، پشتیبانی رسمی را از این نوع داده به عمل نمی‌آورند؛ اما با توجه به سورس باز بودن Entity framework، یک Fork مستقل از آن تهیه شده‌است و این نوع داده‌ی جدید به همراه متدهای مرتبط با آن، به این Fork اضافه شده‌اند.
- اصل Fork در اینجا
- تاریخچه‌ی این Fork غیر رسمی در اینجا
- بسته‌ی نیوگت آن در اینجا

چون تیم EF در نگارش فعلی این کتابخانه حاضر به افزودن این نوع جدید نشده‌است، بنابراین بجای بسته‌ی اصلی Entity framework نیاز است بسته‌ی EntityFrameworkWithHierarchyId را نصب کنید.

```
PM> install-package EntityFrameworkWithHierarchyId
```

یک تذکر مهم:
چون امضای دیجیتال این بسته، با امضای دیجیتال بسته‌ی اصلی EF یکی نیست، اگر پروژه‌ی شما صرفا از EF استفاده می‌کند، مشکلی نخواهید داشت. اما اگر برای مثال از ASP.NET Identity کامپایل شده‌ی برای کار با EF اصلی استفاده کنید، پیام یافت نشدن DLL مرتبط را دریافت خواهید کرد.

تعریف مدلی با خاصیتی از نوع جدید HierarchyId

```
public class Employee
{
    public int Id { get; set; }

    [Required, MaxLength(100)]
    public string Name { get; set; }

    [Required]
    public HierarchyId Node { get; set; } // نوع داده جدید
}
```

در اینجا مدلی را ملاحظه می‌کنید که از نوع داده‌ی جدید HierarchyId استفاده می‌کند. همانطور که عنوان شد این نوع در بسته‌ی EntityFrameworkWithHierarchyId موجود است.

تعریف Context و مقدار دهی اولیه‌ی آن

در این حالت Context برنامه به همراه تنظیمات اولیه‌ی Migrations آن یک چنین شکلی را پیدا خواهد کرد:

```
public class MyContext : DbContext
{
    public DbSet<Employee> Employees { get; set; }

    public MyContext()
        : base("Connection1")
    {
        this.Database.Log = log => Console.WriteLine(log);
    }
}

public class Configuration : DbMigrationsConfiguration<MyContext>
{
    public Configuration()
```

```
    {
        AutomaticMigrationsEnabled = true;
        AutomaticMigrationDataLossAllowed = true;
    }

    protected override void Seed(MyContext context)
    {
        if (context.Employees.Any())
            return;

        context.Database.ExecuteSqlCommand(
            "ALTER TABLE [dbo].[Employees] ADD NodePath as Node.ToString() persisted");
        context.Database.ExecuteSqlCommand(
            "ALTER TABLE [dbo].[Employees] ADD Level AS Node.GetLevel() persisted");
        context.Database.ExecuteSqlCommand(
            "ALTER TABLE [dbo].[Employees] ADD ManagerNode as Node.GetAncestor(1) persisted");
        context.Database.ExecuteSqlCommand(
            "ALTER TABLE [dbo].[Employees] ADD ManagerNodePath as Node.GetAncestor(1).ToString()
persisted");

        context.Database.ExecuteSqlCommand(
            "ALTER TABLE [dbo].[Employees] ADD CONSTRAINT [UK_EmployeeNode] UNIQUE NONCLUSTERED
(Node)");
        context.Database.ExecuteSqlCommand(
            "ALTER TABLE [dbo].[Employees]  WITH CHECK ADD CONSTRAINT [EmployeeManagerNodeNodeFK] " +
            "FOREIGN KEY([ManagerNode]) REFERENCES [dbo].[Employees] ([Node])");

        context.Employees.Add(new Employee { Name = "Root", Node = new HierarchyId("/") });
        context.Employees.Add(new Employee { Name = "Emp1", Node = new HierarchyId("/1/") });
        context.Employees.Add(new Employee { Name = "Emp2", Node = new HierarchyId("/2/") });
        context.Employees.Add(new Employee { Name = "Emp3", Node = new HierarchyId("/1/1/") });
        context.Employees.Add(new Employee { Name = "Emp4", Node = new HierarchyId("/1/1/1/") });
        context.Employees.Add(new Employee { Name = "Emp5", Node = new HierarchyId("/2/1/") });
        context.Employees.Add(new Employee { Name = "Emp6", Node = new HierarchyId("/1/2/") });

        base.Seed(context);
    }
}
```

در اینجا نحوه‌ی تعریف رکوردهای جدید مبتنی بر HierarchyId را مشاهده می‌کنید که توسط آن‌ها تعدادی کارمند، در یک سازمان فرضی ثبت شده‌اند.

همچنین چند فیلد محاسباتی نیز بر اساس امکانات توکار SQL Server اضافه شده‌اند. متدهایی مانند ToString، GetLevel، GetAncestor و امثال آن جزئی از پیاده سازی توکار SQL Server هستند. همچنین این متدها توسط کتابخانه‌ی EntityFrameworkWithHierarchyId نیز ارائه شده‌اند.

کوئری نویسی

مرتب سازی رکوردها بر اساس HierarchyId آن‌ها

```
using (var context = new MyContext())
{
    Console.WriteLine("\ngetItems OrderByDescending(employee => employee.Node)");

    var employees = context.Employees.OrderByDescending(employee => employee.Node).ToList();
    foreach (var employee in employees)
    {
        Console.WriteLine("{0} {1}", employee.Id, employee.Node);
    }
 }
```

با این خروجی

```
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Name] AS [Name],
    [Extent1].[Node] AS [Node]
    FROM [dbo].[Employees] AS [Extent1]
```

```
    ORDER BY [Extent1].[Node] DESC


6 /2/1/
3 /2/
7 /1/2/
5 /1/1/1/
4 /1/1/
2 /1/
1 /
```

یافتن یک HierarchyId خاص و سپس یافتن کلیه‌ی فرزندان آن در یک سطح پایین‌تر

```
using (var context = new MyContext())
{
    Console.WriteLine("\nGetAncestor(1) of /1/");

    var firstItem = context.Employees.Single(employee => employee.Node == new HierarchyId("/1/"));
    foreach (var item in context.Employees.Where(employee => firstItem.Node ==
employee.Node.GetAncestor(1)))
    {
        Console.WriteLine("{0} {1}", item.Id, item.Name);
    }
}
```

این کوئری را به این شکل نیز می‌توان عنوان کرد: یافتن یک HierarchyId و سپس یافتن کلیه نودهایی که والدشان (GetAncestor) این HierarchyId است. عدد یک در اینجا مشخص کننده‌ی Level یا سطح است.
با این خروجی:

```
SELECT TOP (2)
    [Extent1].[Id] AS [Id],
    [Extent1].[Name] AS [Name],
    [Extent1].[Node] AS [Node]
    FROM [dbo].[Employees] AS [Extent1]
    WHERE cast('/1/' as hierarchyid) = [Extent1].[Node]

SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Name] AS [Name],
    [Extent1].[Node] AS [Node]
    FROM [dbo].[Employees] AS [Extent1]
    WHERE (@p__linq__0 = ([Extent1].[Node].GetAncestor(1))) OR ((@p__linq__0 IS
NULL) AND ([Extent1].[Node].GetAncestor(1) IS NULL))
-- p__linq__0: '/1/' (Type = Object)

4 Emp3
7 Emp6
```

کوئری‌های فوق را می‌توان بجای استفاده از متد GetAncestor، با استفاده از متد IsDescendantOf به شکل زیر نیز نوشت:

```
var list = context.Employees.Where(
          employee => employee.Node.IsDescendantOf(new HierarchyId("/1/")) &&
                              employee.Node.GetLevel() == 2).ToList();
```

با این خروجی SQL (یک کوئری بجای دو کوئری):

```
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Name] AS [Name],
    [Extent1].[Node] AS [Node]
    FROM [dbo].[Employees] AS [Extent1]
    WHERE (([Extent1].[Node].IsDescendantOf(cast('/1/' as hierarchyid))) = 1)
    AND (2 = ([Extent1].[Node].GetLevel()))
```

جابجا کردن نودها توسط متد GetReparentedValue

در کوئری ذیل، تمامی فرزندان ریشه‌ی /1/ یافت شده و سپس والد آن‌ها به صورت پویا تغییر داده می‌شود:

```
var items = context.Employees.Where(employee => employee.Node.IsDescendantOf(new HierarchyId("/1/")))
    .Select(employee => new
    {
        Id = employee.Id,
        OrigPath = employee.Node,
        ReparentedValue = employee.Node.GetReparentedValue(new HierarchyId("/1/"),
HierarchyId.GetRoot()),
        Level = employee.Node.GetLevel()
    }).ToList();

foreach (var item in items)
{
    Console.WriteLine("Id:{0}; OrigPath:{1}; ReparentedValue:{2}; Level:{3}", item.Id, item.OrigPath,
item.ReparentedValue, item.Level);
}
```

با این خروجی

```
SELECT
    [Extent1].[Id] AS [Id],
    [Extent1].[Node] AS [Node],
    [Extent1].[Node].GetReparentedValue(cast('/1/' as hierarchyid), hierarchyid::GetRoot()) AS [C1],
    [Extent1].[Node].GetLevel() AS [C2]
    FROM [dbo].[Employees] AS [Extent1]
    WHERE ([Extent1].[Node].IsDescendantOf(cast('/1/' as hierarchyid))) = 1


Id:2; OrigPath:/1/; ReparentedValue:/; Level:1
Id:4; OrigPath:/1/1/; ReparentedValue:/1/; Level:2
Id:5; OrigPath:/1/1/1/; ReparentedValue:/1/1/; Level:3
Id:7; OrigPath:/1/2/; ReparentedValue:/2/; Level:2
```

کدهای کامل این مثال را از اینجا می‌توانید دریافت کنید

HierarcyIdTests.zip