


در [قسمت قبل](#) ساختار اصلی و پیاده‌سازی ابتدایی یک پرووایدر سفارشی دیتابیزی شرح داده شد. در این قسمت ادامه بحث و مطالب پیشرفته‌تر آورده شده است.

تولید یک پرووایدر منابع دیتابیزی - بخش دوم

در بخش دوم این سری مطلب، ساختار دیتابیس و مباحث پیشرفته پیاده‌سازی کلاس‌های نشان داده‌شده در بخش اول در [قسمت قبل](#) شرح داده می‌شود. این مباحث شامل نحوه کش صحیح و بهینه داده‌های دریافتی از دیتابیس، پیاده‌سازی فرایند fallback، و پیاده‌سازی مناسب کلاس DbResourceManager برای مدیریت کل عملیات است.

ساختار دیتابیس

برای پیاده‌سازی منابع دیتابیزی روش‌های مختلفی برای آرایش جداول جهت ذخیره انواع ورودی‌ها می‌توان در نظر گرفت. مثلاً در صورتی که حجم و تعداد منابع بسیار باشد و نیز منابع دیتابیزی به اندازه کافی در دسترس باشد، می‌توان به ازای هر منبع یک جدول در نظر گرفت. یا در صورتی که منابع داده‌ای محدودتر باشند می‌توان به ازای هر کالچر یک جدول در نظر گرفت و تمام منابع مربوط به یک کالچر را درون یک جدول ذخیره کرد. در هر صورت نحوه انتخاب آرایش جداول منابع کاملاً بستگی به شرایط کاری و سلايق برنامه‌نویسی دارد. برای مطلب جاری به عنوان یک راه‌حل ساده و کارآمد برای پروژه‌های کوچک و متوسط، تمام ورودی‌های منابع درون یک جدول با ساختاری مانند زیر ذخیره می‌شود:

	Column Name	Data Type	Allow Nulls
	Id	bigint	<input type="checkbox"/>
	Name	nvarchar(200)	<input type="checkbox"/>
	[Key]	nvarchar(200)	<input type="checkbox"/>
	Culture	nvarchar(6)	<input type="checkbox"/>
	Value	nvarchar(MAX)	<input type="checkbox"/>

نام این جدول را با در نظر گرفتن شرایط موجود می‌توان Resources گذاشت.

ستون Name برای ذخیره نام منبع در نظر گرفته شده است. این نام برابر نام منابع درخواستی در سیستم مدیریت منابع ASP.NET است که در واقع برابر همان نام فایل منبع اما بدون پسوند .resx است.

ستون Key برای نگهداری کلید ورودی منبع استفاده می‌شود که دقیقاً برابر همان مقداری است که درون فایل‌های .resx ذخیره می‌شود.

ستون Culture برای ذخیره کالچر ورودی منبع به کار می‌رود. این مقدار می‌تواند برای کالچر پیش‌فرض برنامه برابر رشته خالی باشد.

ستون Value نیز برای نگهداری مقدار ورودی منبع استفاده می‌شود.

برای ستون Id می‌توان از GUID نیز استفاده کرد. در اینجا برای راحتی کار از نوع داده bigint و خاصیت Identity برای تولید خودکار آن در Sql Server استفاده شده است.

نکته: برای امنیت بیشتر می‌توان یک Unique Constraint بر روی سه فیلد Name و Key و Culture اعمال کرد.

برای نمونه به تصویر زیر که ذخیره تعدادی ورودی منبع را درون جدول Resources نمایش می‌دهد دقت کنید:

Id	Name	Key	Culture	Value
1	GlobalTexts	Yes		yesssss
2	GlobalTexts	Yes	fa	بله
3	GlobalTexts	Yes	fr	oui
4	GlobalTexts	No		no
5	GlobalTexts	No	fa	خیر
6	GlobalTexts	No	fr	pas
7	Default.aspx	Label1.Text		Hello
10	Default.aspx	Label1.ForeColor		red
11	Default.aspx	Label1.Text	en-US	hello
13	Default.aspx	Label1.ForeColor	en-US	blue
14	Default.aspx	Label1.Text	fa	درود
16	Default.aspx	Label1.ForeColor	fa	red
17	Default.aspx	Label2.Text		GoodBye
18	Default.aspx	Label2.ForeColor		orange
19	Default.aspx	Label2.Text	en-US	goodbye
20	Default.aspx	Label2.ForeColor	en-US	green
21	dir 1/page 1.aspx	Label1.Text		sssss
22	dir 1/page 1.aspx	Label2.Text		aaaaa
23	dir 1/page 1.aspx	Label1.Text	en-US	String 1
24	dir 1/page 1.aspx	Label2.Text	en-US	String 2
25	dir 1/page 1.aspx	Label1.Text	fa	رشته 1
26	dir 1/page 1.aspx	Label2.Text	fa	رشته 2

اصلاح کلاس DbResourceProviderFactory

برای ذخیره منابع محلی، جهت اطمینان از یکسان بودن نام منبع، متد مربوطه در کلاس DbResourceProviderFactory باید به صورت زیر تغییر کند:

```
public override IResourceProvider CreateLocalResourceProvider(string virtualPath)
{
    if (!string.IsNullOrEmpty(virtualPath))
    {
        virtualPath = virtualPath.Remove(0, virtualPath.IndexOf('/') + 1); // removes everything from start to the first '/'
    }
    return new LocalDbResourceProvider(virtualPath);
}
```

با این تغییر مسیرهای درخواستی چون "~/Default.aspx" و یا "/Default.aspx" هر دو به صورت "Default.aspx" در می آیند تا با نام ذخیره شده در دیتابیس یکسان شوند.

ارتباط با دیتابیس

خوشبختانه برای تبادل اطلاعات با جدول بالا امروزه راه های زیادی وجود دارد. برای پیاده سازی آن مثلا می توان از یک اینترفیس استفاده کرد. سپس با استفاده از سازوکارهای موجود مثلا به کارگیری [IoC](#)، نمونه مناسبی از پیاده سازی اینترفیس مذکور را در اختیار برنامه قرار داد. اما برای جلوگیری از پیچیدگی بیش از حد و دور شدن از مبحث اصلی، برای پیاده سازی فعلی از EF Code First به صورت مستقیم در پروژه استفاده شده است که [سری آموزشی کاملی از آن](#) در همین سایت وجود دارد.

پس از پیاده سازی کلاس های مرتبط برای استفاده از EF Code First، از کلاس ResourceData که در بخش اول نیز نشان داده شده بود، برای کپسوله کردن ارتباط با داده ها استفاده می شود که نمونه ای ابتدایی از آن در زیر آورده شده است:

```
using System.Collections.Generic;
using System.Linq;
using DbResourceProvider.Models;

namespace DbResourceProvider.Data
{
    public class ResourceData
    {
        private readonly string _resourceName;
        public ResourceData(string resourceName)
        {
            _resourceName = resourceName;
        }
        public Resource GetResource(string resourceKey, string culture)
        {
            using (var data = new TestContext())
            {
                return data.Resources.SingleOrDefault(r => r.Name == _resourceName && r.Key == resourceKey && r.Culture == culture);
            }
        }
        public List<Resource> GetResources(string culture)
        {
            using (var data = new TestContext())
            {
                return data.Resources.Where(r => r.Name == _resourceName && r.Culture == culture).ToList();
            }
        }
    }
}
```

کلاس فوق نسبت به نمونه ای که در قسمت قبل نشان داده شد کمی فرق دارد. بدین صورت که برای راحتی بیشتر نام منبع

درخواستی به جای پارامتر متدها، در اینجا به عنوان پارامتر کانستراکتور وارد می‌شود.

نکته: در صورتی که این کلاس‌ها در پروژه‌ای جداگانه قرار دارند، باید `ConnectionString` مربوطه در فایل کانفیگ برنامه مقصد نیز تنظیم شود.

کش کردن ورودی‌ها

برای کش کردن ورودی‌ها این نکته را که قبلاً هم به آن اشاره شده بود باید در نظر داشت:

پس از اولین درخواست برای هر منبع، نمونه تولیدشده از پرووایدر مربوطه در حافظه سرور کش خواهد شد.

یعنی متدهای کلاس `DbResourceProviderFactory` به‌ازای هر منبع تنها یکبار فراخوانی می‌شود. نمونه‌های کش‌شده از پروایدرهای کلی و محلی به همراه تمام محتویاتشان (مثلاً نمونه تولیدی از کلاس `DbResourceManager`) تا زمان `Unload` شدن سایت در حافظه سرور باقی می‌مانند. بنابراین عملیات کشینگ ورودی‌ها را می‌توان درون خود کلاس `DbResourceManager` به‌ازای هر منبع انجام داد.

برای کش کردن ورودی‌های هر منبع می‌توان چند روش را درپیش گرفت. روش اول این است که به‌ازای هر کلید درخواستی تنها ورودی مربوطه از دیتابیس فراخوانی شده و در برنامه کش شود. این روش برای حالاتی که تعداد ورودی‌ها یا تعداد درخواست‌های کلیدهای هر منبع کم باشد مناسب خواهد بود.

یکی از پیاده‌سازی این روش این است که ورودی‌ها به‌ازای هر کالچر ذخیره شوند. پیاده‌سازی اولیه این نوع فرایند کشینگ در کلاس `DbResourceManager` به صورت زیر است:

```
using System.Collections.Generic;
using System.Globalization;
using DbResourceProvider.Data;
namespace DbResourceProvider
{
    public class DbResourceManager
    {
        private readonly string _resourceName;
        private readonly Dictionary<string, Dictionary<string, object>> _resourceCacheByCulture;
        public DbResourceManager(string resourceName)
        {
            _resourceName = resourceName;
            _resourceCacheByCulture = new Dictionary<string, Dictionary<string, object>>();
        }
        public object GetObject(string resourceKey, CultureInfo culture)
        {
            return GetCachedObject(resourceKey, culture.Name);
        }
        private object GetCachedObject(string resourceKey, string cultureName)
        {
            if (!_resourceCacheByCulture.ContainsKey(cultureName))
                _resourceCacheByCulture.Add(cultureName, new Dictionary<string, object>());
            var cachedResource = _resourceCacheByCulture[cultureName];
            lock (this)
            {
                if (!cachedResource.ContainsKey(resourceKey))
                {
                    var data = new ResourceData(_resourceName);
                    var dbResource = data.GetResource(resourceKey, cultureName);
                    if (dbResource == null) return null;
                    var cachedResources = _resourceCacheByCulture[cultureName];
                    cachedResources.Add(dbResource.Key, dbResource.Value);
                }
            }
            return cachedResource[resourceKey];
        }
    }
}
```

همانطور که قبلاً توضیح داده شد کش پرووایدرهای منابع به‌ازای هر منبع درخواستی (و به تبع آن نمونه‌های موجود در آن مثل `DbResourceManager`) برعهده خود ASP.NET است. بنابراین برای کش کردن ورودی‌های درخواستی هر منبع در کلاس `DbResourceManager` تنها کافی است آن‌ها را درون یک متغیر محلی در سطح کلاس (فیلد) ذخیره کرد. کاری که در کد بالا در متغیر `_resourceCacheByCulture` انجام شده است. در این متغیر که از نوع دیکشنری تعریف شده است کلیدهای هر عضو آن برابر نام کالچر مربوطه است. مقادیر هر عضو این دیکشنری نیز خود یک دیکشنری است که ورودی‌های منابع مربوط به کالچر مربوطه در

آن ذخیره می‌شوند.

عملیات در متد `GetCachedObject` انجام می‌شود. همان‌طور که می‌بینید ابتدا وجود ورودی موردنظر در متغیر کشینگ بررسی می‌شود و در صورت عدم وجود، مقدار آن مستقیماً از دیتابیس درخواست می‌شود. سپس این مقدار درخواستی ابتدا درون متغیر کشینگ ذخیره شده (به همراه بلاک `lock`) و در نهایت برگشت داده می‌شود.

نکته: کل فرایند بررسی وجود کلید در متغیر کشینگ (شرط دوم در متد `GetCachedObject`) درون بلاک `lock` قرار داده شده است تا در درخواست‌های همزمان احتمال افزودن چندباره یک کلید از بین برود.

پیاده‌سازی دیگر این فرایند کشینگ، ذخیره ورودی‌ها براساس نام کلید به جای نام کالچر است. یعنی کلید دیکشنری اصلی نام کلید و کلید دیکشنری داخلی نام کالچر است که این روش زیاد جالب نیست.

روش دوم که بیشتر برای برنامه‌های بزرگ با ورودی‌ها و درخواست‌های زیاد به کار می‌رود این است که در هر بار درخواست به دیتابیس به جای دریافت تنها همان ورودی درخواستی، تمام ورودی‌های منبع و کالچر درخواستی استخراج شده و کش می‌شود تا تعداد درخواست‌های به سمت دیتابیس کاهش یابد. برای پیاده‌سازی این روش کافی است تغییرات زیر در متد `GetCachedObject` اعمال شود:

```
private object GetCachedObject(string resourceKey, string cultureName)
{
    lock (this)
    {
        if (!_resourceCacheByCulture.ContainsKey(cultureName))
        {
            _resourceCacheByCulture.Add(cultureName, new Dictionary<string, object>());
            var cachedResources = _resourceCacheByCulture[cultureName];
            var data = new ResourceData(_resourceName);
            var dbResources = data.GetResources(cultureName);
            foreach (var dbResource in dbResources)
            {
                cachedResources.Add(dbResource.Key, dbResource.Value);
            }
        }
    }
    var cachedResource = _resourceCacheByCulture[cultureName];
    return !cachedResource.ContainsKey(resourceKey) ? null : cachedResource[resourceKey];
}
```

در اینجا هم می‌توان به جای استفاده از نام کالچر برای کلید دیکشنری اصلی از نام کلید ورودی منبع استفاده کرد که چندان توصیه نمی‌شود.

نکته: انتخاب یکی از دو روش فوق برای فرایند کشینگ کاملاً به شرایط موجود و سلیقه برنامه نویس بستگی دارد.

فرایند Fallback

درباره فرایند `fallback` به اندازه کافی در قسمت‌های قبلی توضیح داده شده است. برای پیاده‌سازی این فرایند ابتدا باید به نوعی به سلسله مراتب کالچرهای موجود از کالچر جاری تا کالچر اصلی و پیش فرض سیستم دسترسی پیدا کرد. برای اینکار ابتدا باید با استفاده از روشی کالچر والد یک کالچر را بدست آورد. کالچر والد کالچری است که عمومیت بیشتری نسبت به کالچر موردنظر دارد. مثلاً کالچر `fa`، کالچر والد `fa-IR` است. همچنین کالچر `Invariant` به عنوان والد تمام کالچرها شناخته می‌شود. خوشبختانه در کلاس `CultureInfo` (که در قسمت‌های قبلی شرح داده شده است) یک پراپرتی با عنوان `Parent` وجود دارد که کالچر والد را برمی‌گرداند.

برای رسیدن به سلسله مراتب مذکور در کلاس `ResourceManager` دات نت، از کلاسی با عنوان `ResourceFallbackManager` استفاده می‌شود. هرچند این کلاس با سطح دسترسی `internal` تعریف شده است اما نام‌گذاری نامناسبی دارد زیرا کاری که می‌کند به عنوان `Manager` هیچ ربطی ندارد. این کلاس با استفاده از یک کالچر ورودی، یک `enumerator` از سلسله مراتب کالچرها که در بالا صحبت شد تهیه می‌کند.

با استفاده پیاده‌سازی موجود در کلاس `ResourceFallbackManager` کلاسی با عنوان `CultureFallbackProvider` تهیه کردم که به صورت زیر است:

```

using System.Collections;
using System.Collections.Generic;
using System.Globalization;
namespace DbResourceProvider
{
    public class CultureFallbackProvider : IEnumerable<CultureInfo>
    {
        private readonly CultureInfo _startingCulture;
        private readonly CultureInfo _neutralCulture;
        private readonly bool _tryParentCulture;
        public CultureFallbackProvider(CultureInfo startingCulture = null,
                                      CultureInfo neutralCulture = null,
                                      bool tryParentCulture = true)
        {
            _startingCulture = startingCulture ?? CultureInfo.CurrentCulture;
            _neutralCulture = neutralCulture;
            _tryParentCulture = tryParentCulture;
        }
        #region Implementation of IEnumerable<CultureInfo>
        public IEnumerator<CultureInfo> GetEnumerator()
        {
            var reachedNeutralCulture = false;
            var currentCulture = _startingCulture;
            do
            {
                if (_neutralCulture != null && currentCulture.Name == _neutralCulture.Name)
                {
                    yield return CultureInfo.InvariantCulture;
                    reachedNeutralCulture = true;
                    break;
                }
                yield return currentCulture;
                currentCulture = currentCulture.Parent;
            } while (_tryParentCulture && !HasInvariantCultureName(currentCulture));
            if (!_tryParentCulture || HasInvariantCultureName(_startingCulture) || reachedNeutralCulture)
                yield break;
            yield return CultureInfo.InvariantCulture;
        }
        #endregion
        #region Implementation of IEnumerable
        IEnumerator IEnumerable.GetEnumerator()
        {
            return GetEnumerator();
        }
        #endregion
        private bool HasInvariantCultureName(CultureInfo culture)
        {
            return culture.Name == CultureInfo.InvariantCulture.Name;
        }
    }
}

```

این کلاس که اینترفیس `IEnumerable<CultureInfo>` را پیاده‌سازی کرده است، سه پارامتر کانستراکتور دارد. اولین پارامتر، کالچر جاری یا آغازین را مشخص می‌کند. این کالچری است که تولید `enumerator` مربوطه از آن آغاز می‌شود. در صورتی که این پارامتر نال باشد مقدار کالچر UI در ثرد جاری برای آن در نظر گرفته می‌شود. مقدار پیش‌فرضی که برای این پارامتر در نظر گرفته شده است، `null` است. پارامتر بعدی کالچر خنثی موردنظر کاربر است. این کالچری است که در صورت رسیدن `enumerator` به آن کار پایان خواهد یافت. در واقع کالچر پایانی `enumerator` است. این پارامتر می‌تواند نال باشد. مقدار پیش‌فرضی که برای این پارامتر در نظر گرفته شده است، `null` است. پارامتر آخر هم تعیین می‌کند که آیا `enumerator` از کالچرهای والد استفاده بکند یا خیر. مقدار پیش‌فرضی که برای این پارامتر در نظر گرفته شده است، `true` است. کار اصلی کلاس فوق در متد `GetEnumerator` انجام می‌شود. در این کلاس یک حلقه `do-while` وجود دارد که `enumerator` را با استفاده از کلمه کلیدی `yield` تولید می‌کند. در این متد ابتدا در صورت نال نبودن کالچر خنثی ورودی، بررسی می‌شود که آیا نام کالچر جاری حلقه (که در متغیر محلی `currentCulture` ذخیره شده است) برابر نام کالچر خنثی است یا خیر. در صورت برقراری شرط، کار این حلقه با برگشت `CultureInfo.InvariantCulture` پایان می‌یابد. کالچر بدون زبان و فرهنگ و موقعیت مکانی است که در واقع به عنوان کالچر والد تمام کالچرها در نظر گرفته می‌شود. پراپرتی `Name` این کالچر برابر `string.Empty` است.

کار حلقه با برگشت مقدار کالچر جاری enumerator ادامه می‌یابد. سپس کالچر جاری با کالچر والدش مقداردهی می‌شود. شرط قسمت while حلقه تعیین می‌کند که در صورتی که کلاس برای استفاده از کالچرهای والد تنظیم شده باشد، تا زمانی که نام کالچر جاری برابر نام کالچر Invariant نباشد، تولید اعضای enumerator ادامه یابد.

در انتها نیز در صورتی که با شرایط موجود، قبلا کالچر Invariant برگشت داده نشده باشد این کالچر نیز yield می‌شود. در واقع در صورتی که استفاده از کالچرهای والد اجازه داده نشده باشد یا کالچر آغازین برابر کالچر Invariant باشد و یا قبلا به دلیل رسیدن به کالچر خنثی ورودی، مقدار کالچر Invariant برگشت داده شده باشد، enumerator قطع شده و عملیات پایان می‌یابد. در غیر این صورت کالچر Invariant به عنوان کالچر پایانی برگشت داده می‌شود.

استفاده از CultureFallbackProvider

با استفاده از کلاس CultureFallbackProvider می‌توان عملیات جستجوی ورودی‌های درخواستی را با ترتیبی مناسب بین تمام کالچرهای موجود به انجام رسانید.

برای استفاده از این کلاس باید تغییراتی در متد GetObject کلاس DbResourceManager به صورت زیر اعمال کرد:

```
public object GetObject(string resourceKey, CultureInfo culture)
{
    foreach (var currentCulture in new CultureFallbackProvider(culture))
    {
        var value = GetCachedObject(resourceKey, currentCulture.Name);
        if (value != null) return value;
    }
    throw new KeyNotFoundException("The specified 'resourceKey' not found.");
}
```

با استفاده از یک حلقه foreach درون enumerator کلاس CultureFallbackProvider، کالچرهای مورد نیاز برای fallback یافته می‌شوند. در اینجا از مقادیر پیش فرض دو پارامتر دیگر کانستراکتور کلاس CultureFallbackProvider استفاده شده است. سپس به ازای هر کالچر یافته شده مقدار ورودی درخواستی بدست آمده و در صورتی که نال نباشد (یعنی ورودی مورد نظر برای کالچر جاری یافته شود) آن مقدار برگشت داده می‌شود و در صورتی که نال باشد عملیات برای کالچر بعدی ادامه می‌یابد. در صورتی که ورودی درخواستی یافته نشود (خروج از حلقه بدون برگشت مقداری برای ورودی منبع درخواستی) استثنای KeyNotFoundException صادر می‌شود تا کاربر را از اشتباه رخ داده مطلع سازد.

آزمایش پرووایدر سفارشی

ابتدا تنظیمات مورد نیاز فایل کانفیگ را که در [قسمت قبل](#) نشان داده شد، در برنامه خود اعمال کنید.

داده‌های نمونه نشان داده شده در ابتدای این مطلب را در نظر بگیرید. حال اگر در یک برنامه وب اپلیکیشن، صفحه Default.aspx در ریشه سایت حاوی دو کنترل زیر باشد:

```
<asp:Label ID="Label1" runat="server" meta:resourcekey="Label1" />
<asp:Label ID="Label2" runat="server" meta:resourcekey="Label2" />
```

خروجی برای کالچر "en-US" (معمولا پیش فرض، اگر تنظیمات سیستم عامل تغییر نکرده باشد) چیزی شبیه تصویر زیر خواهد بود:

hello goodbye

سپس تغییر زیر را در فایل web.config اعمال کنید تا کالچر UI سایت به fa تغییر یابد (به بخش uiCulture="fa" دقت کنید):

```
<globalization uiCulture="fa" resourceProviderFactoryType =
```

```
"DbResourceProvider.DbResourceProviderFactory, DbResourceProvider" />
```

بنابراین صفحه Default.aspx با همان داده‌های نشان داده شده در بالا به صورت زیر تغییر خواهد کرد:

GoodBye درود

می‌بینید که با توجه به عدم وجود مقداری برای Label12.Text برای کالچر fa، عملیات fallback اتفاق افتاده است.

بحث و نتیجه‌گیری

کار تولید یک پرووایدر منابع سفارشی دیتابسی به اتمام رسید. تا اینجا اصول کلی تولید یک پرووایدر سفارشی شرح داده شد. بدین ترتیب می‌توان برای هر حالت خاص دیگری نیز پرووایدرهای سفارشی مخصوص ساخت تا مدیریت منابع به آسانی تحت کنترل برنامه نویسی قرار گیرد.

اما نکته‌ای را که باید به آن توجه کنید این است که در پیاده‌سازی‌های نشان داده شده با توجه به نحوه کش‌شدن مقادیر ورودی‌ها، اگر این مقادیر در دیتابیس تغییر کنند، تا زمانیکه سایت ریست نشود این تغییرات در برنامه اعمال نخواهد شد. زیرا همانطور که اشاره شد، مدیریت نمونه‌های تولیدشده از پرووایدرهای منابع برای هر منبع درخواستی در نهایت برعهده ASP.NET است. بنابراین باید مکانیزمی پیاده شود تا کلاس DbResourceManager از به‌روزرسانی ورودی‌های کش‌شده اطلاع یابد تا آنها را ریفرش کند.

در ادامه درباره روش‌های مختلف نحوه پیاده‌سازی قابلیت به‌روزرسانی ورودی‌های منابع در زمان اجرا با استفاده از پرووایدرهای منابع سفارشی بحث خواهد شد. همچنین راه‌حل‌های مختلف استفاده از این پرووایدرهای سفارشی در جاهای مختلف پروژه‌های MVC شرح داده می‌شود.

البته مباحث پیشرفته‌تری چون تزریق وابستگی برای پیاده‌سازی لایه ارتباط با دیتابیس در بیرون و یا تولید یک Factory برای تزریق کامل پرووایدر منابع از بیرون نیز جای بحث و بررسی دارد.

منابع

<http://weblogs.asp.net/thangchung/archive/2010/06/25/extending-resource-provider-for-soring-resources-in-the-database.aspx>

<http://msdn.microsoft.com/en-us/library/aa905797.aspx>

<http://msdn.microsoft.com/en-us/library/system.web.compilation.resourceproviderfactory.aspx>

<http://www.dotnetframework.org/default.aspx/.../ResourceFallbackManager@cs>

<http://www.codeproject.com/Articles/14190/ASP-NET-2-0-Custom-SQL-Server-ResourceProvider>

<http://www.west-wind.com/presentations/wwdbresourceprovider>

نظرات خوانندگان

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۳/۰۸ ۰:۴۲

با تشکر از کار زیبای شما
لطفاً برچسب [resource](#) را اضافه کنید تا پیوستگی مطالب حفظ شود.

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱:۴۰

با تشکر از دقت نظر شما.
برچسب Resource هم اضافه شد.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۳/۰۸ ۳:۱۵

مهندس بک سوال؟
مشکلی نداره ما سه جدول:
1- جدولی برای ذخیره نام کالچرها
2- جدولی برای ذخیره عنوان کلیدهای اصلی
3- جدولی برای ذخیره مقادیر یک کالچر برای یک کلید خاص

تعریف کنیم؟
اگر درست فهمیده باشیم فقط باید بخش بازیابی کلیدها تغییر کنه درسته؟

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۳/۰۸ ۸:۴۱

اون وقت حداقل 2 تا join باید بنویسید و وجود هر join یعنی کمتر شدن سرعت دسترسی به اطلاعات. چرا؟ چه تکرار اطلاعاتی رو مشاهده می‌کنید که قصد دارید تا این حد نرمالش کنید؟ نام و کلید و فرهنگ یک موجودیت هستند.

نویسنده: یوسف نژاد
تاریخ: ۱۳۹۲/۰۳/۰۸ ۹:۱۱

دلیل خاصی برای تفکیک این چینی وجود نداره و همونطور که دوستمون گفتن این روشی که شما اشاره کردین مشکلات و معایبی هم به همراه داره.
روش اشاره شده تو این مطلب تو بیش از 99 درصد پروژه‌ها کفایت میکنه. فقط تو پروژه‌های بسیار بسیار بزرگ با ورودی‌های منابع بسیار زیاد (چند صد هزار و یا بیشتر) تغییر این ساختار برای رسیدن به کارایی مناسب میتونه مفید باشه.
در هر صورت اگر نیاز به تغییر ساختار جدول دارین فقط لایه دسترسی به بانک باید تغییر بکنه و فرایند کلی دسترسی به ورودی‌های منابع ذخیره شده در دیتابیس باید به همون صورتی باشه که در اینجا آورده شده. یعنی در نهایت با استفاده از سه پارامتر نام منبع، نام کالچر و عنوان کلید درخواستی کار استخراج مقدار ورودی باید انجام بشه.

نویسنده: صابر فتح الهی
تاریخ: ۱۳۹۲/۰۳/۰۸ ۱۰:۱۴

برای طراحی یک سامانه مدیریت محتوا با کلی مازول فکر می‌کنم حرفم منطقی باشه مهندس، در ضمن همونجوری که مهندس [یوسف نژاد](#) فرمودن اطلاعات در بازیابی اولیه کش میشه و تا ری ستارت شدن سایت در حافظه می‌مونه، فکر می‌کنم چندان تاثیری

بروی کارایی داشته باشه با توجه به فرضیات، فرض کن من 10000 عنوان دارم، 30 تا زبان دارم در این صورت توی یک جدول زبان انگلیسی (en-کالچر انگلیسی) 10000 بار تکرار میشه علاوه بر اون عنوان مثلا "نام کاربری" به ازای 30 زبان 30 بار تکرار میشه زیادم حرف من غیر منطقی نیست و الا حرف شما درسته بله join سرعت پایین میاره اما ما که قرار نیست زیادی دسترسی به این جداول داشته باشیم.

"پس از اولین درخواست برای هر منبع، نمونه تولیدشده از پرووایدر مربوطه در حافظه سرور کش خواهد شد." سخن مهندس

[یوسف نژاد](#)

نویسنده: محسن خان

تاریخ: ۱۳۹۲/۰۳/۰۸ ۱۲:۱۰

یک سری از برآوردها خیلی هستند. حتی میکروسافت هم با لشگر مترجم‌هایی که داره مثلا برای شیرپوینت تجاری خودش زیر 10 تا زبان رو تونسته ارائه بده.

نویسنده: بهنام حقی

تاریخ: ۱۳۹۳/۰۱/۳۱ ۱۷:۰۹

با سلام

من این حالت رو میخوام با uow میخوام پیاده سازی کنم. میخوام یک سری تغییرات تو ساختار جدول بدم. یک جدول برای مدیریت اضافه و حذف زبان (نام، RTL، ISO، Culture و ...) و جدول دیگم برای ریسورس‌ها (کلید، اسم، مقدار) در واقع میخوام مقادیر ریسورس‌ها با اضافه و حذف شدن یک زبان به سیستم مدیریت بشه. میخوام ببینم که چه پیشنهادی برای این حالت دارید؟