Tipo : Enunciado de laboratorio
Capítulo : Servicios Windows Communication Foundation (WCF)
Duración : 60 minutos

I. OBJETIVO

Modificar el UnitOfWork.

II. REQUISITOS

Los siguientes elementos de software son necesarios para la realización del laboratorio:

- Windows 8.1
- SQL Server 2016
- Visual Studio 2017

III. EJECUCIÓN DEL LABORATORIO

– Ejercicio 7.1: Agregar un nuevo método para la lectura de un procedimiento almacenado y mapear la información.

1. Abrir SQL Management Studio y crear el siguiente procedimiento almacenado:

```
CREATE PROCEDURE dbo.CustomerInvoice
	@invoiceId int,
	@email nvarchar(60)
AS
BEGIN
	SET NOCOUNT ON;

	SELECT
	inv.BillingCountry,
	line.Quantity,
	line.UnitPrice,
	inv.Total,
	cust.FirstName,
	cust.LastName,
	cust.Email
	FROM dbo.Invoice inv
	INNER JOIN dbo.InvoiceLine line on line.InvoiceId=inv.InvoiceId
	INNER JOIN dbo.Customer cust on cust.CustomerId=inv.CustomerId
	WHERE cust.Email=@email AND line.InvoiceId=@invoiceId
END
GO
```

2. Abrir la solución del capítulo anterior.

3. En el proyecto **Models**, creamos la clase: **CustomerInvoice.cs** que debe de tener las siguientes propiedades:

```
namespace Models
{
    public class CustomerInvoice
    {
```

```
        public string BillingCountry { get; set; }
        public int Quantity { get; set; }
        public decimal UnitPrice { get; set; }
        public decimal Total { get; set; }
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Email { get; set; }
    }
}
```

4. En el proyecto de **DataAccess**, refactorizamos el código para evitar la duplicidad de código**.**

   – Editamos la clase ArtistRepository de la siguiente manera:

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

namespace DataAccess.Repositories
{
    public class ArtistRepository : Repository<Artist>,
IArtistRepository
    {
        public ArtistRepository(ChinookContext context) :
base(context)
        {
        }
        public IEnumerable<Artist> GetListArtistByStore()
        {
            return
ChinookContext.Database.SqlQuery<Artist>("GetListOfArtist");
        }

        public Artist GetByName(string name)
        {
            return ChinookContext.Artists.FirstOrDefault(artist =>
artist.Name == name);
        }

    }
}
```

   – Refactorizamos la clase Repository.cs

```
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;

namespace DataAccess.Repositories
{
    public class Repository<TEntity> : IRepository<TEntity> where
TEntity : class
    {
        protected readonly DbContext Context;
        protected ChinookContext ChinookContext
        {
            get { return Context as ChinookContext; }
        }
        public Repository(DbContext context)
        {
            Context = context;
        }
```

```
            public void Add(TEntity entity)
            {
                Context.Set<TEntity>().Add(entity);
            }

            public TEntity GetById(int id)
            {
                return Context.Set<TEntity>().Find(id);
            }

            public IEnumerable<TEntity> GetAll()
            {
                return Context.Set<TEntity>().ToList();
            }

            public void Remove(TEntity entity)
            {
                Context.Set<TEntity>().Remove(entity);
            }

            public int Count()
            {
                return Context.Set<TEntity>().Count();
            }
        }
    }
```

5. En el proyecto de **DataAccess**, en la carpeta **Repositories** crear la interfaz: **ICustomerRepository.cs** con el siguiente código:

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

namespace DataAccess.Repositories
{
    public interface ICustomerRepository: IRepository<Customer>
    {
        IEnumerable<CustomerInvoice> CustomerInvoice(string customerEmail,
int invoiceCode );
    }
}
```

6. En la misma carpeta **Repositories**, crear la clase: **CustomerRepository.cs** que debe de tener el siguiente código:

```
using Models;
using System.Collections.Generic;
using System.Data.SqlClient;

namespace DataAccess.Repositories
{
    public class CustomerRepository : Repository<Customer>,
ICustomerRepository
    {
        public CustomerRepository(ChinookContext context) : base(context)
        {
        }
        public IEnumerable<CustomerInvoice> CustomerInvoice(string
customerEmail, int invoiceCode)
        {
            var email = new SqlParameter("@email", customerEmail);
            var invoiceId = new SqlParameter("@invoiceId", invoiceCode);
```

```
            return
ChinookContext.Database.SqlQuery<CustomerInvoice>("dbo.CustomerInvoice
@invoiceId, @email", invoiceId, email);
        }
    }
}
```

7. Procedemos a editar la clase: **UnitOfWork.cs**

```
using DataAccess.Repositories;
using Models;

namespace DataAccess
{
    public class UnitOfWork : IUnitOfWork
    {
        private readonly ChinookContext _context;

        public UnitOfWork(ChinookContext context)
        {
            _context = context;
            Artists = new ArtistRepository(_context);
            Albums = new Repository<Album>(_context);
            Customers = new CustomerRepository(_context);
            Employees = new Repository<Employee>(_context);
            Genres = new Repository<Genre>(_context);
            Invoices = new Repository<Invoice>(_context);
            InvoiceLines = new Repository<InvoiceLine>(_context);
            MediaTypes = new Repository<MediaType>(_context);
            Playlists = new Repository<Playlist>(_context);
            Tracks = new Repository<Track>(_context);
        }

        public IArtistRepository Artists { get; private set; }
        public Repository<Album> Albums { get; private set; }
        public ICustomerRepository Customers { get; private set; }
        public Repository<Employee> Employees { get; private set; }
        public Repository<Genre> Genres { get; private set; }
        public Repository<Invoice> Invoices { get; private set; }
        public Repository<InvoiceLine> InvoiceLines { get; private set; }
        public Repository<MediaType> MediaTypes { get; private set; }
        public Repository<Playlist> Playlists { get; private set; }
        public Repository<Track> Tracks { get; private set; }

        public int Complete()
        {
            return _context.SaveChanges();
        }

        public void Dispose()
        {
            _context.Dispose();
        }
    }
}
```

8. Para poder validar que nuestro cambio funcione adecuadamente creamos un nuevo test en el proyecto **DataAccess.Test**, cuyo nombre será **CustomerRepositoryTest.cs,** agregar el siguiente código:

```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using System.Linq;
```

```
namespace DataAccess.Test
{
    [TestClass]
    public class CustomerRepositoryTest
    {
        private readonly UnitOfWork _unitOfWork;
        public CustomerRepositoryTest()
        {
            _unitOfWork = new UnitOfWork(new ChinookContext());
        }

        [TestMethod]
        public void Get_List_Of_Customer_Invoice()
        {
            var results =
_unitOfWork.Customers.CustomerInvoice("leonekohler@surfeu.de",
1).ToList();
            Assert.AreEqual(results[0].Email, "leonekohler@surfeu.de");
            Assert.AreEqual(results[1].Email, "leonekohler@surfeu.de");
            Assert.AreEqual(results.Count() > 0, true);
        }
    }
}
```

9. Ejecutar el test y el estado debe de ser **Passed**.

IV. EVALUACIÓN

1. ¿Por qué implementar un servicio WCF?

   Windows Communication Foundation (WCF) es un marco de trabajo para la creación de aplicaciones orientadas a servicios. Con WCF, es posible enviar datos como mensajes asincrónicos de un extremo de servicio a otro. Un extremo de servicio puede formar parte de un servicio disponible continuamente hospedado por IIS, o puede ser un servicio hospedado en una aplicación. Un extremo puede ser un cliente de un servicio que solicita datos de un extremo de servicio. Los mensajes pueden ser tan simples como un carácter o una palabra que se envía como XML, o tan complejos como una secuencia de datos binarios. A continuación, se indican unos cuantos escenarios de ejemplo:

   - Un servicio seguro para procesar transacciones comerciales.
   - Un servicio que proporciona datos actualizados a otras personas, como un informe sobre tráfico u otro servicio de supervisión.
   - Un servicio de chat que permite a dos personas comunicarse o intercambiar datos en tiempo real.
   - Una aplicación de panel que sondea los datos de uno o varios servicios y los muestra en una presentación lógica.
   - Exponer un flujo de trabajo implementado utilizando Windows Workflow Foundation como un servicio WCF.

   Si bien era posible crear tales aplicaciones antes de que existiera WCF, con WCF el desarrollo de extremos resulta más sencillo que nunca. En resumen, WCF se ha diseñado para ofrecer un enfoque manejable para la creación de servicios web y clientes de servicios web.