
NHIBERNATE

baufest

Sobre el instructor

Diego Tubello

dtubello@baufest.com

Technical Expert

13+ años de experiencia en desarrollo de Software

baufest

baufest

Objetivos del curso

- Inicialmente existirá una breve introducción que permitirá incorporar conceptos que se utilizarán a lo largo del curso, para luego, ir evolucionando en el conocimiento del framework
- Se aclara que el curso será exclusivamente sobre **NHibernate**, descartando entrar en detalle de otros conceptos de arquitectura en capas o desarrollo Full Stack, sobre los que por supuesto habrá menciones pero no se entrará en detalle
- El curso irá evolucionando día a día hasta poder completar un ejercicio integrador final

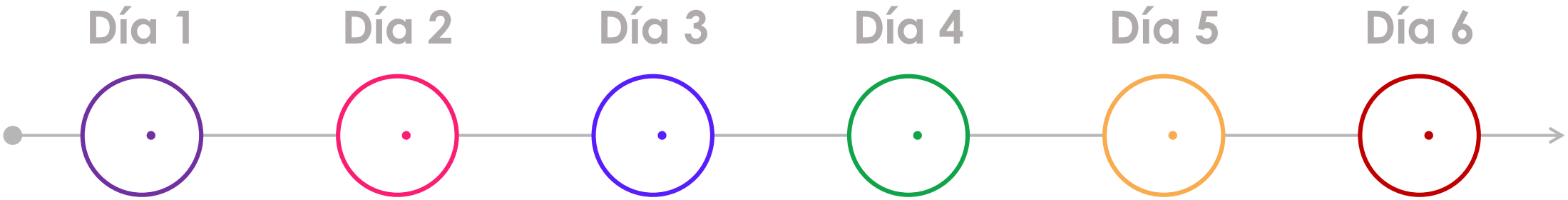
baufest

Organización del curso

Organización del curso (1)

Temario por día

baufest



Día 1

- Presentación
- Introducción
- Mapeo básico
- Entorno de desarrollo

Día 2

- Arquitectura
- Configuración
- Session
- Operaciones CRUD

Día 3

- Convenciones
- Mapeo de claves
- Mapeo de relaciones
- Otros mapeos

Día 4

- Transacciones
- Operaciones CRUD con modelo completo
- Revisión de SQL generado
- Consultas

Día 5

- SQL Nativo
- Buenas prácticas
- Uso en aplicaciones Web

Día 6

- Ejercicio integrador
- Dudas del curso
- Despedida

Organización del curso (2)

Eventos en común para todos los días

baufest

Coffee Break

Resolución de dudas de los días anteriores

baufest

Día 1 – Objetivos del día

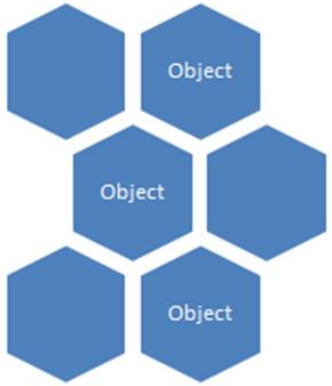
- Introducir conceptos y actividades que serán utilizados a lo largo del curso
- Explicar que es un ORM, que problema resuelve
- Realizar un primer acercamiento a **NHibernate**
  - Introducción teórica
  - Mapeo básico
  - Práctica: Utilización de entorno de trabajo y un “Hola Mundo”

Conceptos preliminares

Diferentes modelos

baufest

- Modelo de Objetos
  - Objetos
  - Propiedades
  - Agregación, Composición, Herencia



Objects in Memory

- Modelo Relacional
  - Tablas
  - Campos
  - Registros
  - Relaciones por Foreign Keys

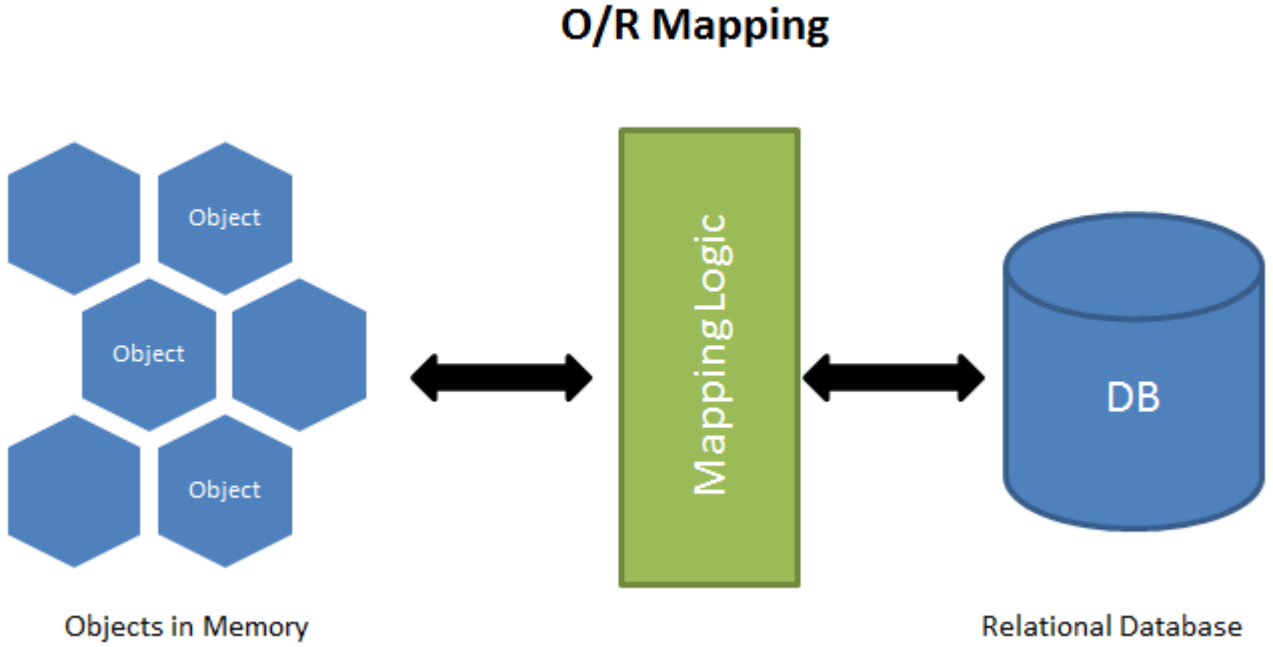
Relational Database

Conceptos preliminares

ORM: Object/Relational Mapper

baufest

- Permite mapear un modelo de objetos a un modelo relacional.
- Se mapea un conjunto de objetos que colaboran entre si a un conjunto de tablas relacionadas.
- Permite persistir objetos de una forma “transparente”
- Se manipulan los datos como si tuviéramos colecciones de objetos en memoria



Object/Relational Mapper

¿Por qué usar un ORM?

baufest

- Permite modelar con objetos abstrayéndonos del modelo de datos relacional.
- Se reducen los tiempos de desarrollo al no tener que escribir código de base de datos.
- Se manipulan los “datos” como si tuviéramos colecciones de objetos en memoria.
- Se facilitan los *refactors*, ya que al modificar el modelo de objetos el ORM adapta todas las consultas automáticamente eliminando la necesidad de modificar manualmente consultas y stored procedures.
- El esquema de carga de objetos de de la base suele ser muy configuragle: cache, lazy/eager loading, batch
- Soporte para múltiples motores de bases de datos y versiones (por configuración)

Abstraerse de la base de datos no implica olvidar que existe

Object/Relational Mapper

Implementaciones

baufest

Object/Relational Mapper

¿Por qué NHibernate?

baufest

- Nació en 2006 como un port del framework Hibernate para Java, pero con el tiempo evolucionó de forma distinta
- Es open-source (LGPL)
- Soporta la mayoría de los motores de bases de datos y sus versiones
- Es flexible y posee muchos puntos de extensión
- Sigue activamente en desarrollo
- Soporte para .Net Core a partir de la versión 5

Mapeo de objetos

Alternativas: Xml, Fluent, Auto-mappings

baufest

Archivos XML

- Mecanismo original de mapeo portado de Hibernate (de la época dorada del XML)

Fluent NHibernate

- Surge como un proyecto separado con la idea de genera mapeos type-safe
- Permite hacer mapeos manuales o “automáticos”
- Convention over configuration

Mapeo de objetos

Objetos

baufest

NHibernate es capaz de mapear objetos “comunes” (POCO):

- No se requiere implementar una interface y extender una clase
- Tampoco deben tener ningún código relacionado a NH

Solo requiere dos cosas:

- La clase debe tener un constructor por defecto sin parámetros (o no tener ningún constructor)
- Todos los métodos y propiedades de la clase deben ser *virtual*

```
namespace Intro.NHibernate.Entities
{
    public class Product
    {
        public virtual int Id { get; set; }
        public virtual string Code { get; set; }
        public virtual string Description { get; set; }
        public virtual decimal RetailPrice { get; set; }
        public virtual Category Category { get; set; }
    }
}
```

Mapeo de objetos

XML

baufest

El mapeo de objetos puede definirse en un archivo xml de la siguiente manera.

Todos los objetos deben tener un atributo como identificador.

```xml
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2"
  assembly="Intro.NHibernate"
  namespace="Intro.NHibernate.Entities" default-lazy="false">

  <class name="Product">
    <id name="Id">
      <generator class="identity" />
    </id>
    <property name="Code" />
    <property name="Description" />
    <property name="RetailPrice" />
    <many-to-one name="Category" column="Category_id" />
  </class>

</hibernate-mapping>
```

```csharp
namespace Intro.NHibernate.Entities
{
    public class Product
    {
        public virtual int Id { get; set; }
        public virtual string Code { get; set; }
        public virtual string Description { get; set; }
        public virtual decimal RetailPrice { get; set; }
        public virtual Category Category { get; set; }
    }
}
```

Mapeo de objetos

Fluent Nhibernate: Fluent mappings

baufest

- Ofrece una alternativa al mapeo de archivos xml.
- En lugar de documentos xml, se escriben los mapeos en código fuertemente tipado.
- Al ser mapeos tipados, facilita la refactorización de código y hace el mapeo mas legible.

```
public class ProductMap : ClassMap<Product>
{
    public ProductMap()
    {
        Id(x => x.Id).GeneratedBy.Identity();
        Map(x => x.Code);
        Map(x => x.Description);
        Map(x => x.RetailPrice);
        References(x => x.Category).Cascade.None();
    }
}
```

```
namespace Intro.NHibernate.Entities
{
    public class Product
    {
        public virtual int Id { get; set; }
        public virtual string Code { get; set; }
        public virtual string Description { get; set; }
        public virtual decimal RetailPrice { get; set; }
        public virtual Category Category { get; set; }
    }
}
```

Mapeo de objetos

Fluent Nhibernate: auto-mappings

baufest

- Permite mapear un modelo de objetos sin escribir código de mapeo, con muy poco código.
- Se base en el concepto de convenciones sobre configuración (convention over configuration)
- Requiere que nuestro modelo respete ciertas convenciones
- Si las convenciones por defecto no se adaptan a nuestras convenciones, podemos definir y utilizar nuestras propias convenciones
- Para modificar mapeos puntuales tiene el concepto de overrides

Entorno de desarrollo

¿Qué herramientas vamos a utilizar?

baufest

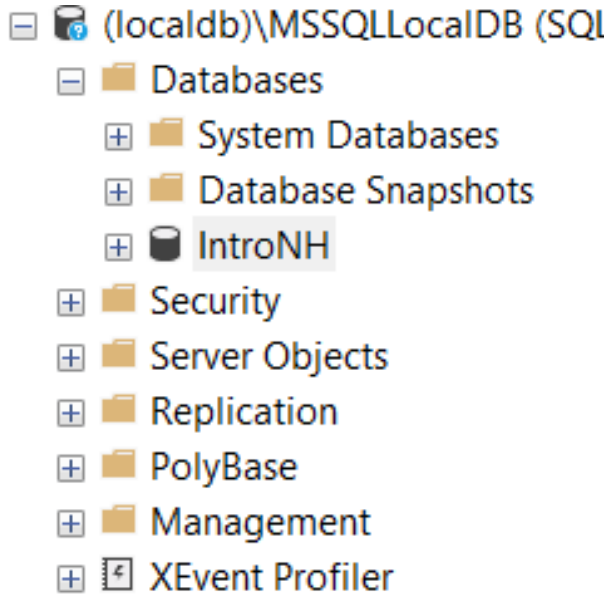
- Visual Studio
- SQL Server Management Studio
- SQL Server Profiler

Práctica

¡Hola Mundo!

baufest

- En SQL Management Studio
  - Conectarse a la base de datos local
  - Crear una nueva base de datos con el nombre “IntroNH”

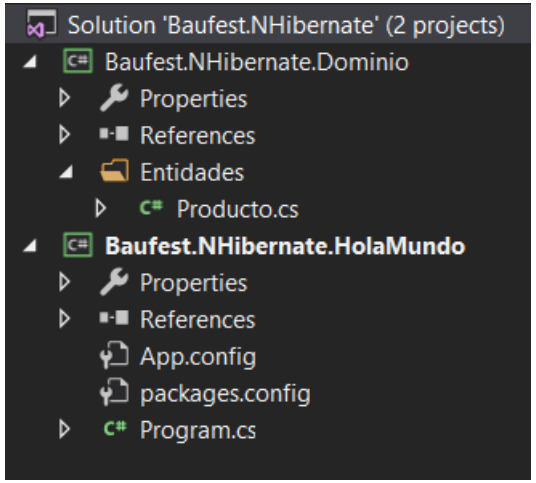


Práctica

¡Hola Mundo!

baufest

- En Visual Studio
  - Crear una nueva solución del tipo blank solution con el nombre “Baufest.Nhibernate”
  - Agregar un proyecto del tipo “Console Application” con el nombre “Baufest.NHibernate.HolaMundo”
  - Agregar un nuevo proyecto del tipo “Class Library” con el nombre “Baufest.Nhibernate.Dominio”
    - Eliminar el archivo Class1.cs
    - Agregar la carpeta “Entidades”
    - Agregar la clase “Producto” dentro de la carpeta Entidades
  - Instalar los paquetes NuGet en el proyecto HolaMundo
    - NHibernate
    - FluentNHibernate




```
0 references
public class Producto
{
    0 references
    public virtual int Id { get; set; }

    0 references
    public virtual string Nombre { get; set; }

    0 references
    public virtual string Descripcion { get; set; }

    0 references
    public virtual decimal Precio { get; set; }
}
```

Práctica

¡Hola Mundo!

baufest

- En Visual Studio
  - En la clase Program crear el método “CrearSessionFactory” con la configuración de NHibernate
  - Dentro del método Main crear un producto y guardarlo en la base de datos
  - Ejecutar el proyecto!

```
public static ISessionFactory CrearSessionFactory()
{
    return Fluently
          .Configure()
          .Database(
             MsSqlConfiguration.MsSql2012.ConnectionString(
                      x => x.FromConnectionStringWithKey("BaufestNH")))
          .Mappings(m => m.AutoMappings.Add(
                     AutoMap.AssemblyOf<Producto>()
                         .Where(t => t.Namespace == typeof(Producto).Namespace)))
          .ExposeConfiguration(cfg => new SchemaUpdate(cfg).Execute(false, true))
          .BuildSessionFactory();
}
```

```
static void Main(string[] args)
{
    var sessionfactory = CrearSessionFactory();
    using(var session = sessionfactory.OpenSession())
    {
        var producto = new Producto
        {
            Nombre = "Lenovo T470",
            Descripcion = "Laptop Lenovo T470 Core i5, 16GB RAM, SSD 128GB",
            Precio = 500
        };

        session.Save(producto);
    }
}
```

baufest

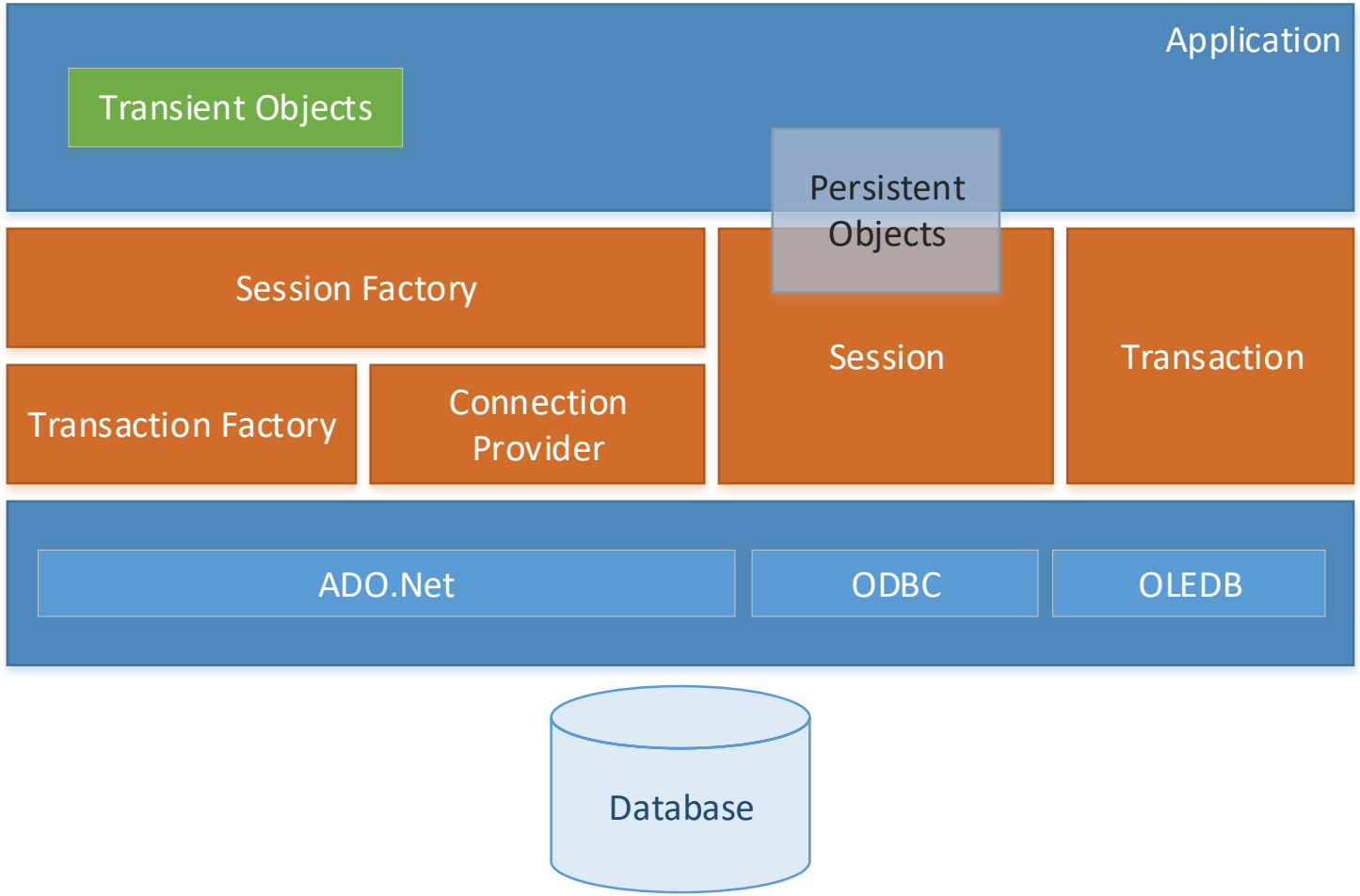
Día 2 – Objetivos del día

- Dudas o consultas de la clase anterior
- Introducir los conceptos de la arquitectura de NHibernate
- Comprender la configuración con sus diferentes opciones
- Comprender el manejo de la sesión
- Realizar operaciones CRUD básicas

Arquitectura

Componentes principales

baufest



Arquitectura

Componentes principales

baufest

- **ISessionFactory**: es una cache con toda la configuración y mapeos “compilados”. Es la Factory para crear las ISession. Se usa una instancia por aplicación.
- **ISession**: representa la conversación entre la aplicación y la base de datos. Se usa una instancia por “operación de negocio” (unit of work). Tiene un tiempo de vida corto.
- **Persistent Objects**: Son objetos que contienen estado persistente (una correlación don datos de la base) y funciones de negocio. Pueden ser POCOs pero están asociados a una única sesión.
- **Transient Objects**: son objetos que no están actualmente asociados a una sesión. Su estado aún no está persistido y no se se persistirá salvo que los asociemos a una sesión y se conviertan en “persistent”.
- **ITransaction**: representa una unidad de trabajo atómica. Es una abstracción de la transacción ADO.Net subyacente.

Configuración

ISessionFactory

baufest

```
Fluently
 .Configure()
 .Database(...) // Configuración de la base y versión
 .Mappings(...) // Configuración de mapeos
 .ExposeConfiguration(...) // Configuraciones adicionales (opcional)
 .BuildSessionFactory();
```

```
public static ISessionFactory CrearSessionFactory()
{
    return Fluently
          .Configure()
          .Database(
             MsSqlConfiguration.MsSql2012.ConnectionString(
                      x => x.FromConnectionStringWithKey("BaufestNH")))
          .Mappings(m => m.AutoMappings.Add(
                      AutoMap.AssemblyOf<Producto>()
                          .Where(t => t.Namespace == typeof(Producto).Namespace)))
          .ExposeConfiguration(cfg => new SchemaUpdate(cfg).Execute(false, true))
          .BuildSessionFactory();
}
```

Configuración

ISessionFactory

baufest

Database permite configurar el motor de base de datos a utilizar:

- **Motor/Versión**: NHibernate soporta múltiples motores de bases de datos en casi todas sus versiones
- **Connection String**: Se puede especificar el connection string o la clave con la que se lo configuró en el web.config o app.config.
- **Driver**: permite configurar el driver del motor de base de datos a utilizar (opcional)
- **Dialect**: permite configurar un dialecto de SQL específico del motor y versión a utilizar (opcional).

```
return Fluently
    .Configure()
    .Database(
      MsSqlConfiguration.MsSql2012
          .Dialect<MsSqlAzure2008Dialect>()
          .ConnectionString(x => x.FromConnectionStringWithKey("IntroNH")))
```

Configuración

ISessionFactory

baufest

Mappings permite configurar los mapeos

- Se puede especificar automapping, las convenciones, overrides y el conjunto de clases a mapear

```
.Mappings(m => m.AutoMappings.Add(
          AutoMap.AssemblyOf<Producto>()
              .Where(t => t.Namespace == typeof(Producto).Namespace)
              .Conventions.AddFromAssemblyOf<ForeignKeyNameConvention>()
              .UseOverridesFromAssemblyOf<VentaMappingOverride>()
              ))
```

- También se pueden especificar los “fluent mappings” en el caso de querer hacerlos manualmente

```
.Mappings(m => m.FluentMappings.AddFromAssemblyOf<ProductoMap>())
```

**Ejemplo*

Configuración

ISessionFactory

baufest

ExposeConfiguration permite hacer configuraciones adicionales

- Habilitar la generación del modelo de la base
- Opciones de quoting de nombres de columnas tablas
- Modificar la forma de generar las consultas
- Configuración de event listeners
- Configuraciones de ADO.Net
- Conexiones, transacciones, cache, logging, etc.

```
.ExposeConfiguration(cfg =>
{
    cfg.SetProperty(Environment.Hbm2ddlKeyWords, "auto-quote");
    cfg.SetProperty(Environment.ConnectionDriver, typeof(SqlClientDriver).AssemblyQualifiedName);
    cfg.LinqToHqlGeneratorsRegistry<MyLinqtoHqlGeneratorsRegistry>();
})
.ExposeConfiguration(RegisterEventListeners)
.ExposeConfiguration(BuildSchema)
.BuildSessionFactory();
```

ISession

Operaciones

baufest

El objeto a través del cual se hace la manipulación de objetos persistentes es la ISession.

Funciona como un contenedor. Cuando se carga un objeto, se guarda en el contenedor de manera que NHibernate registrará cualquier cambia que se le haga al objeto.

Las operaciones principales son:

- **Get** y **Load**: permiten obtener un único objeto por clave primaria
- **Query**: permite hacer una consulta con LINQ sobre el modelo persistente.
- **Add**: adjunta un objeto a la sesión para que sea persistido.
- **Update**: Re-adjunta un objeto a la sesión para actualizar su estado en el modelo persistente.
- **Delete**: elimina un objeto de la sesión para ser eliminado del modelo persistente.

ISession

Estados de una objeto

baufest

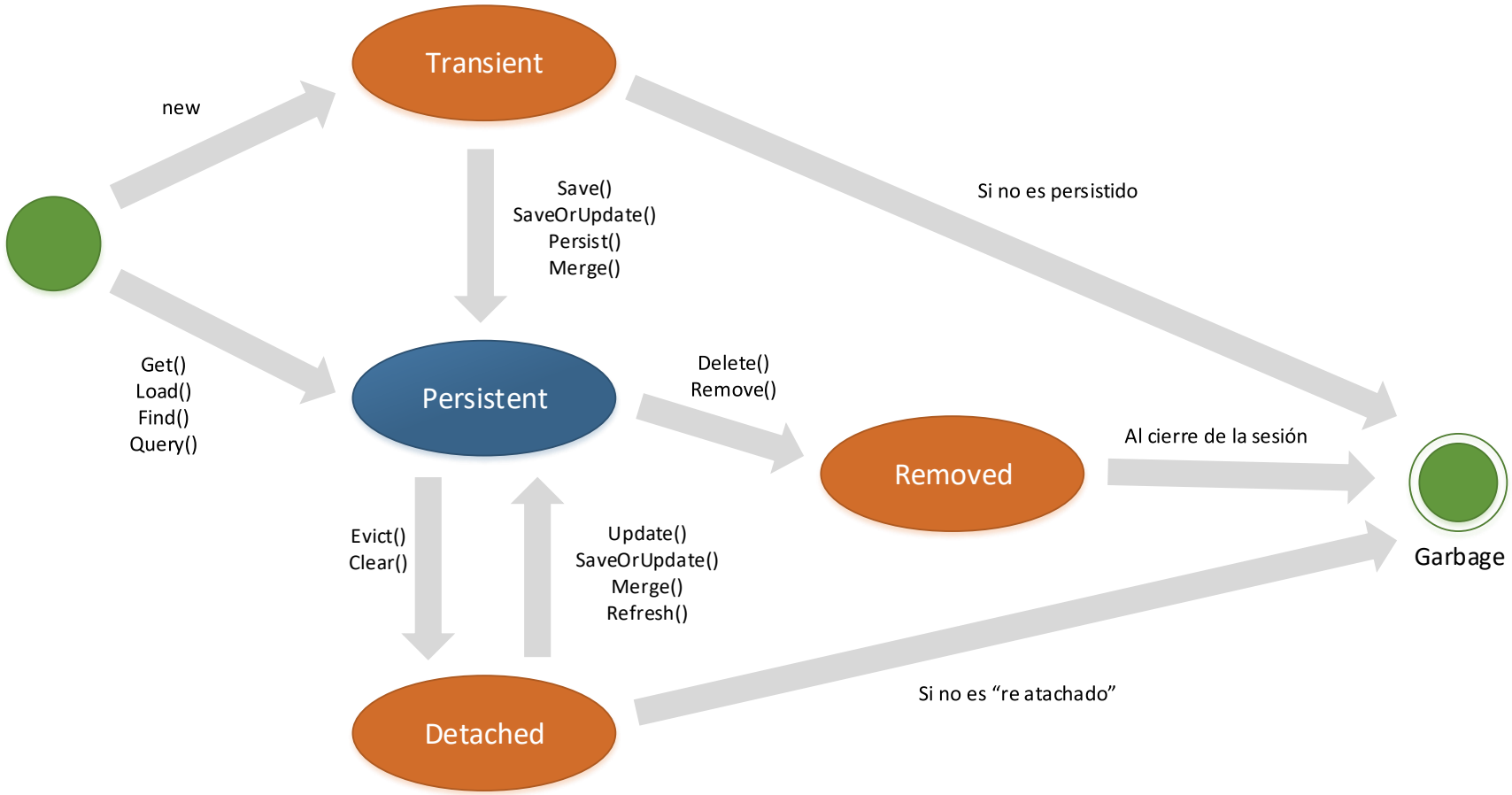
Una instancia de una clase persistente puede pasar por tres diferentes estados con respecto al contexto de persistencia o sesión (ISession).

- **Transient**: el objeto no esta asociado a una sesión y nunca lo estuvo. No tiene un identificador de persistencia (PK)
- **Persistent**: el objeto esta actualmente asociado a una sesión. Tiene un identificador de persistencia (PK), y posiblemente, una registro correspondiente en la base d e datos. Para una sesión particular NHibernate garantiza que dos objetos con la misma clave primaria son el mismo objeto en memoria.
- **Detached**: El objeto estuvo asociado a una sesión.

ISession

Ciclo de vida

baufest



ISession

Flush Mode

baufest

Los cambios que se hacen en la sesión no necesariamente se impactan en la base de datos en ese momento. NHibernate puede diferir un INSERT o UPDATE en la base de datos para agrupar sentencias y optimizar los accesos a la base de datos.

La acción de impactar los cambios de la sesión en la base de datos de llama **Flush**. Se puede configurar cambiando el **FlushMode**:

- **Never**: Los cambios no se sincronizan automáticamente, se deben hace manualmente llamando al método **Flush**
- **Commit**: Los cambios se sincronizan cuando la transacción en curso hace commit
- **Auto**: los flush se hacen automáticamente durante la vida de la sesión para evitar “dirty reads”.
- **Always**: Se hace un Flush automáticamente ante la ejecución de cada consulta

ISession

Flush

baufest

En ocasiones Nhibernate necesita hacer un Flush para sincronizar con la base de datos para mantener la consistencia:

- Cuando se listan registros de la base mediante una consulta
- Cuando se hace commit de una transacción
- Cuando se hace un flush explícito

ISession

Flush

baufest

¿En qué orden se impactan los cambios en un Flush?

1. Todos los inserts, en el mismo orden en que se guardaron usando ISession.Save()
2. Todos las actualizaciones de entidades
3. Todos los borrados de colecciones
4. Todos los inserts y actualizaciones de elementos de las colecciones
5. Todos los inserts de colecciones
6. Borrado de entidades en el mismo orden en que fueron eliminados con ISession.Delete()

Una excepción son las entidades que usan ID identity que se insertan en el momento de ejecutar Save.

ISession

Flush

baufest

Salvo que explícitamente se llame al método Flush() no hay garantías de cuándo la Session ejecuta las llamadas a ADO.Net.

Sin embargo, NHibernate **SI** garantiza:

- El orden en que se ejecutan las sentencias en la base de datos
- Las consultas siempre retornan datos “consistentes” y “actualizados”, nunca devolverán datos incorrectos.

ISession

Lazy Loading

baufest

Cuando se obtiene un objeto de la base de datos, por defecto, NHibernate no carga los objetos relacionados.

Recién obtiene el objeto relacionado al acceder a la propiedad del objetos que se obtuvo en primero lugar.

- En la clase de ejemplo, recién obtendrá el objeto Categoría de la base de datos al invocar *producto.Categoria*

Para que un objeto pueda ser cargado de forma Lazy, todas sus propiedades y métodos deben ser virtual (NH genera proxies dinámicos)

Funciona dentro de la sesión

- Una vez cerrada la sesión ya no se pueden cargar relaciones de forma Lazy

```
public class Producto
{
    2 references
    public virtual int Id { get; set; }

    9 references
    public virtual string Nombre { get; set; }

    5 references
    public virtual string Descripcion { get; set; }

    5 references
    public virtual decimal Precio { get; set; }

    7 references
    public virtual Categoria Categoria { get; set; }
}
```

ISession

Cierre

baufest

Siempre se debe cerrar la sesión para marcar el final de la “conversación” y liberar los recursos ADO.Net.

Se puede hacer explícitamente o con la construcción “using” que hace el Dispose de la Session al salir del scope.

```
var session = sessionfactory.OpenSession();
try
{
    var categoria = new Categoria { Nombre = "Notebooks" };
    session.Save(categoria);
}
finally
{
    session.Close();
}
```

```
using (var session = sessionfactory.OpenSession())
{
    var categoria = new Categoria { Nombre = "Notebooks" };
    session.Save(categoria);
}
```

Práctica

WEB API

baufest

- Descargar la solución de ejemplo de github https://github.com/baufest-ms/Curso-NHibernate-AySA
- Dentro de la carpeta Practica/API hay una solución WEB API base para iniciar los ejercicios
- Recorremos la solución: proyectos, frameworks, dependencias, nswag (swagger)
- Demo de swagger
- Agregar las operaciones CRUD para las entidades que usamos la primera clase
  - Comenzar con la clase Categoria
  - Update detached/attached
  - Hacer lo mismo con la clase Producto. Funciona?
  - DTOs & Lazy Loading

baufest

Día 3 – Objetivos del día

- Dudas o consultas de la clase anterior
- Comprender la forma de modificar los mapeos mediante convenciones y overrides
- Mapeo de claves primarias
- Mapeo de relaciones
- Mapeo con vistas y stored procedures
- Práctica: manejo de objetos con relaciones

Automapping

Mepeo por convenciones

baufest

Fluent NHibernate permite mapear objetos a un modelo relacional sin la necesidad de escribir código o configuración para hacerlo:

- La configuración de mapeos y la estructura de las tablas de la base de datos se infieren a partir del modelo de objetos a mapear
- Esa inferencia la hace en base a una serie de reglas de mapeos predefinidas
- Las clases del modelo a mapear deben respetar algunos patrones y convenciones

Automapping

Convenciones Básicas Estándar

baufest

- Una clase de mapea a una tabla con el mismo nombre
- Las propiedades de una clase de tipos “básicos” de mapean a columnas con los mismos nombres
- Toda clase mapeada debe tener una propiedad Id del tipo *int* que ese mapeará a la columna Id de la tabla y será la primary key auto-incremental *(identity)*
- *Las columnas que representen foreign keys tendrán el nombre:*

  <nombre de la tabla remota>_id

```
public class Product
{
    public virtual int Id { get; set; }
    public virtual string Code { get; set; }
    public virtual string Description { get; set; }
    public virtual decimal RetailPrice { get; set; }
    public virtual Category Category { get; set; }
}
```

```
CREATE TABLE [dbo].[Product](
    [Id] [int] IDENTITY(1,1) NOT NULL,
    [Code] [nvarchar](255) NULL,
    [Description] [nvarchar](255) NULL,
    [RetailPrice] [decimal](19, 5) NULL,
    [Category_id] [int] NULL,
    PRIMARY KEY CLUSTERED ([Id] ASC)
)
```

Automapping

Modificando el estándar

baufest

- Muchas veces las convenciones que trae Fluent NHibernate no se adecúan a las convenciones que ya utilizamos
- Otras veces necesitamos trabajar sobre una base existente que ya viene con convenciones definidas
- El framework tiene dos mecanismos para modificar el comportamiento por defecto:
  - Custom conventions o convenciones de usuario
  - Overrides

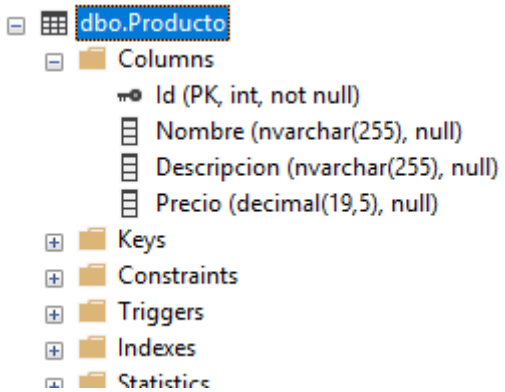
Automapping

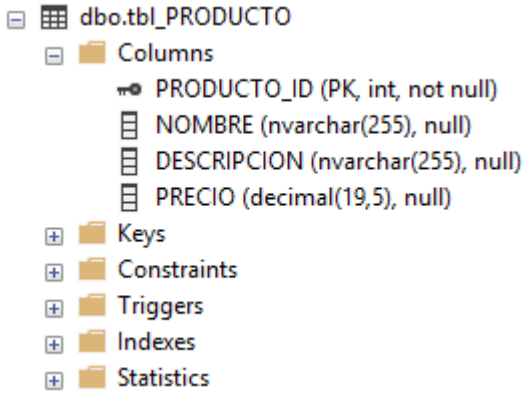
Custom Conventions

baufest

- Permiten cambiar el comportamiento de los mapeos a nivel general. Las principales son:
  - Nombres de las tablas
  - Nombres de las columnas
  - Nombres de las primary keys
  - Nombres de las foreign keys
- Se pueden escribir “inline” en la configuración de la sesión Factory o creando clases que implementan una interface determinada.

La documentación completa se encuentra en https://github.com/FluentNHibernate/fluent-nhibernate/wiki/conventions





Automapping

Overrides

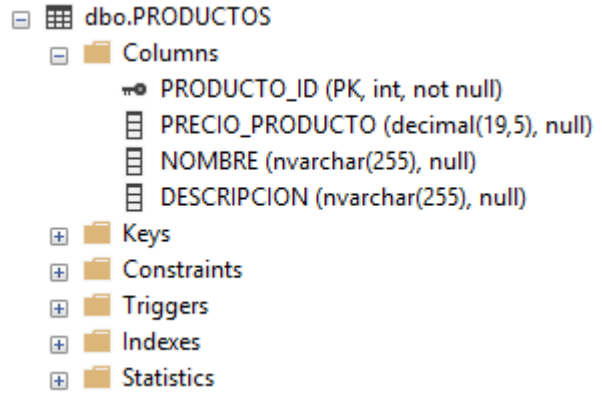
baufest

Permiten cambiar el comportamiento de los mapeos en casos puntuales sin afectar nada más.

Los ejemplos más frecuentes de uso son cuando trabajamos con una base de datos heredada que no puede modificarse y hay tablas que no se ajustan a las convenciones

Se configuran generando una clase que implementa la interface IAutoMappingOverride

```
0 references
public class ProductoMappingOverride : IAutoMappingOverride<Producto>
{
    0 references
    public void Override(AutoMapping<Producto> mapping)
    {
        mapping.Table("PRODUCTOS");
        mapping.Map(x => x.Precio, "PRECIO_PRODUCTO");
    }
}
```



Mapeo de claves primarias

Generación de claves

baufest

NHibernate permite mapear claves primarias de varias formas. El soporte de cada uno de los métodos está condicionado al motor de base de datos que usemos

Las principales son:

- **Identity**: usa una columna del tipo identity auto-incremental en DB2, MySQL y MS SQL. El identificador lo genera la base de datos.
- **Hilo**: usa el algoritmo hi/lo para generar identificadores numéricos
- **Sequence**: usa una se sequence de DB2, PostgreSQL u Oracle.
- **Guid**: usa un System.Guid como identificador
- **Native**: elige identity, sequence o hilo dependiendo de las capacidades del motor
- **Assigned**: la aplicación es responsable de generar y asignar la clave antes de persistir un object.

Mapeo de claves primarias

Claves Compuestas

baufest

NHibernate permite usar claves compuestas que involucren más de una propiedad en caso de ser necesario, con el método de mapping *CompositeId().*

Es muy importante que en estos casos, las clases correspondientes implementen con la misma semántica los métodos *Equals()* y *GetHashCode().*

Este esquema tiene algunas limitaciones ya que el propio objeto es su identificador y ni tenemos in ID “handle” para referirnos a él.

Por ejemplo al usar Load o Get debemos crear un objeto de ejemplo, asignarle las propiedades d e la clave con los valores correspondientes y pasarlo como parámetro al método

Mapeo de relaciones entre objetos

De objetos a tablas

baufest

En el mundo de objetos modelamos las relaciones con:

- Composición
- Agregación
- Herencia

En el mundo relacional modelamos las relaciones con:

- Foreign Keys

¿Cómo unimos estos mundos?

- Existen reglas de conversión de relaciones

Mapeo de relaciones entre objetos

Composición y Agregación

baufest

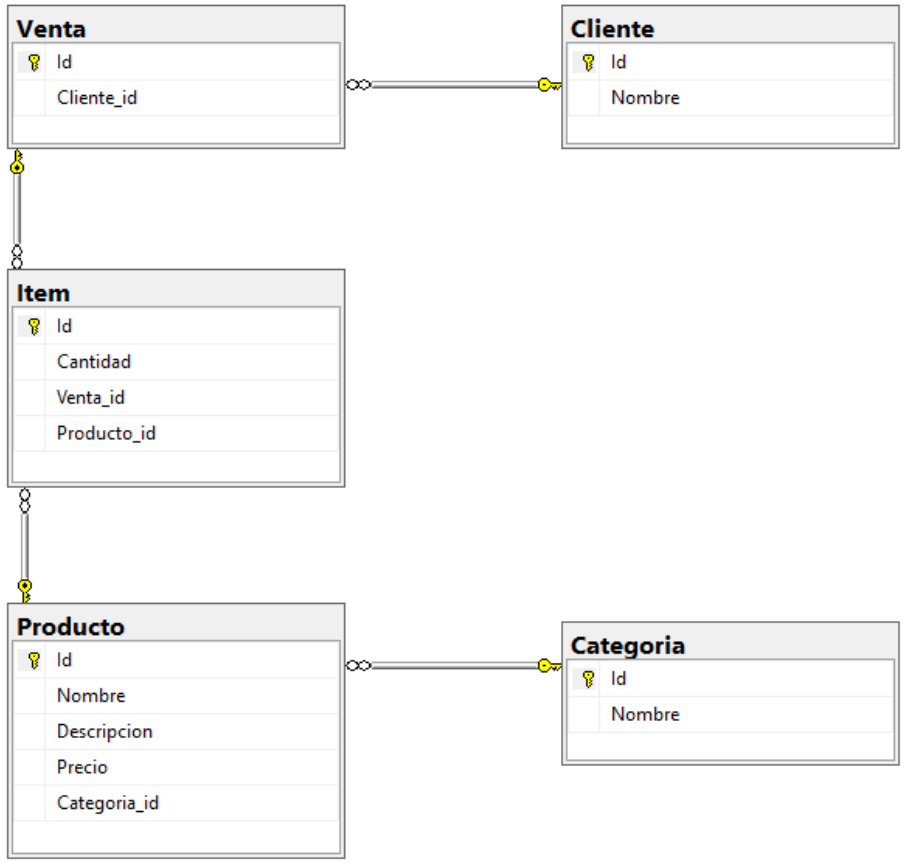
```
public class Venta
{
    0 references
    public virtual int Id { get; set; }

    1 reference
    public virtual Cliente Cliente { get; set; }

    4 references
    public virtual IList<Item> Items { get; set; }
}
```

```
public class Cliente
{
    0 references
    public virtual int Id { get; set; }

    1 reference
    public virtual string Nombre { get; set; }
}
```

```
public class Item
{
    0 references
    public virtual int Id { get; set; }

    1 reference
    public virtual Venta Venta { get; set; }

    1 reference
    public virtual Producto Producto { get; set; }

    1 reference
    public virtual int Cantidad { get; set; }
}
```

```
public class Producto
{
    0 references
    public virtual int Id { get; set; }

    2 references
    public virtual string Nombre { get; set; }

    2 references
    public virtual string Descripcion { get; set; }

    2 references
    public virtual decimal Precio { get; set; }

    2 references
    public virtual Categoria Categoria { get; set; }
}
```

```
public class Categoria
{
    0 references
    public virtual int Id { get; set; }

    2 references
    public virtual string Nombre { get; set; }
}
```



Mapeo de relaciones entre objetos

Composición y Agregación: Cascade

baufest

En la mayor parte de los casos auto-mapping detecta y mapea correctamente estas relaciones sin hacer configuraciones.

En los casos que queramos simular una relación de composición en la que el objeto relacionado es parte del objeto “principal” se hay que modificar la opción de “Cascade”.

El “Cascade” hace que los objetos relacionados se persisten cuando se persiste el objeto que los contiene.

```
public class VentaMappingOverride : IAutoMappingOverride<Venta>
{
    0 references
    public void Override(AutoMapping<Venta> mapping)
    {
        mapping.HasMany(x => x.Items).Cascade.AllDeleteOrphan();
    }
}
```

Mapeo de relaciones entre objetos

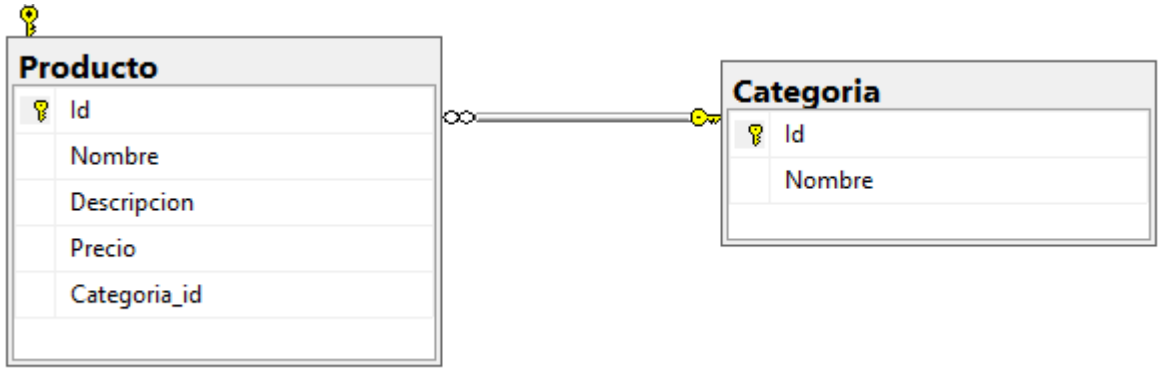
Cardinalidad: many to one

baufest

Es una relación muchos a uno en la que un objeto referencia a otro mediante una propiedad.

En la base de datos se modela como una relación muchos a uno por foreign key.

```
public class Producto
{
    0 references
    public virtual int Id { get; set; }

    2 references
    public virtual string Nombre { get; set; }

    2 references
    public virtual string Descripcion { get; set; }

    2 references
    public virtual decimal Precio { get; set; }

    2 references
    public virtual Categoria Categoria { get; set; }
}
```




```
public class Categoria
{
    0 references
    public virtual int Id { get; set; }

    2 references
    public virtual string Nombre { get; set; }
}
```

Mapeo de relaciones entre objetos

Cardinalidad: one to one

baufest

Es una relación uno a uno en la que un objeto referencia a otro mediante una propiedad. Su uso no es tan frecuente.

En la base se puede modelar de dos formas:

Usando la misma primary key en ambas tablas

Usando una relación por foreign key del tipo uno a muchos pero haciendo que sea unique.

Mapeo de relaciones entre objetos

Cardinalidad: Colecciones, one to many

baufest

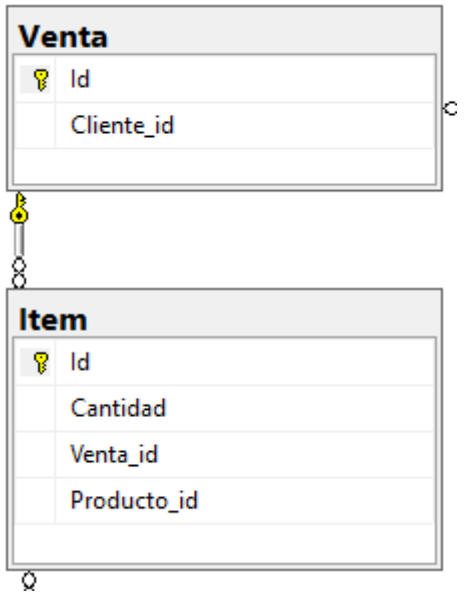
Es una relación uno a muchos en a que un objeto contiene una colección de otros objetos (List, Set, Dictionary, etc.). Es la inversa de la relación many to one.

Se puede tener a nivel objetos una relación bidireccional en la que ambos objetos tienen la referencia al otro

En la base de datos se modela como una relación muchos a uno por foreign key, igual que many to one ya que en un modelo relacional las relaciones no tienen “dirección”.

```
public class Venta
{
    0 references
    public virtual int Id { get; set; }

    1 reference
    public virtual Cliente Cliente { get; set; }

    4 references
    public virtual IList<Item> Items { get; set; }
```

```
public class Item
{
    0 references
    public virtual int Id { get; set; }

    1 reference
    public virtual Venta Venta { get; set; }

    1 reference
    public virtual Producto Producto { get; set; }

    1 reference
    public virtual int Cantidad { get; set; }
}
```



Mapeo de relaciones entre objetos

Cardinalidad: Colecciones, many to many

baufest

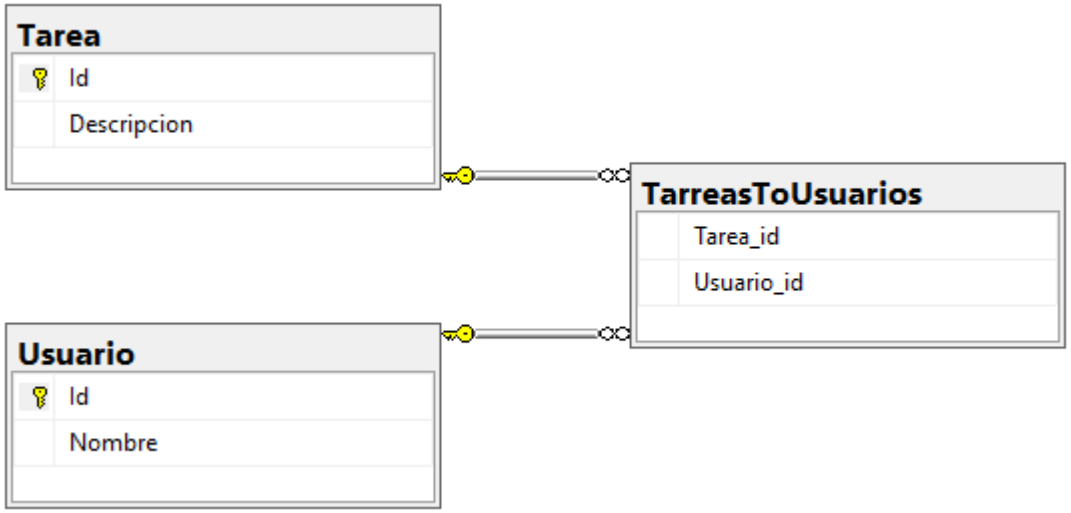
Es una relación muchos a muchos en a que un objeto contiene una colección de otros objetos y del otro lado de la relación sucede lo mismo.

Se puede tener a nivel objetos una relación bidireccional en la que ambos objetos tienen la referencia al otro en una colección.

En la base de datos se modela con una tabla de relaciones:

```
public class Usuario
{
    0 references
    public virtual int Id { get; set; }

    0 references
    public virtual string Nombre { get; set; }

    0 references
    public virtual IList<Tarea> Tarreas { get; set; }
}
```

```
public class Tarea
{
    0 references
    public int Id { get; set; }

    0 references
    public string Descripcion { get; set; }

    0 references
    public IList<Usuario> Usuarios { get; set; }
}
```



Mapeo de relaciones entre objetos

Cardinalidad: Colecciones, many to many (cont)

baufest

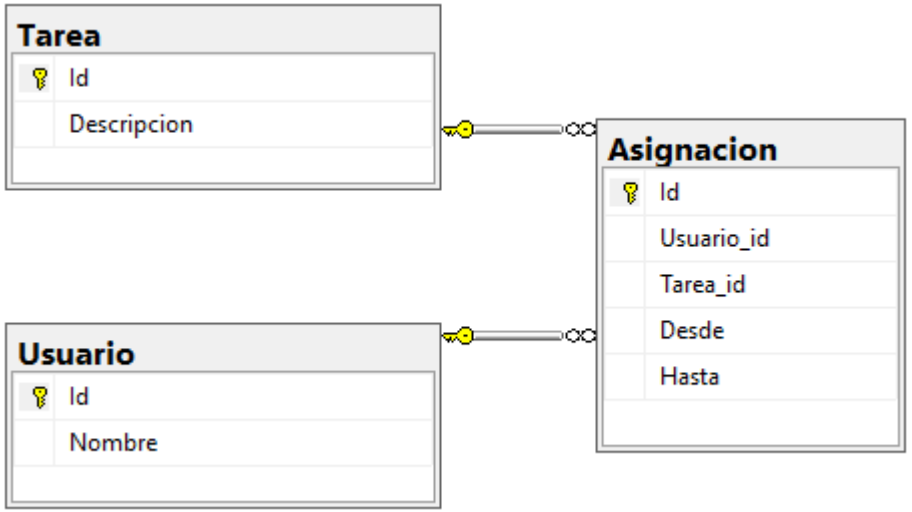
Aunque NHibernate soporte este tipo de relaciones, rara vez es lo correcto. En general una relación many-to-many indica que nos faltó modelar una relación en el modelo de objetos. En la mayor parte de los casos, en la relación necesitamos guardar más información

En el ejemplo anterior podríamos modelar la ”Asignación” y usar relaciones one-to-many / many-to-one. Este esquema además permite agregar, por ejemplo, atributos para modelar las fechas de la asignación.

```
public class Usuario
{
    0 references
    public virtual int Id { get; set; }

    0 references
    public virtual string Nombre { get; set; }

    0 references
    public virtual IList<Asignacion> Asignaciones { get; set; }
}
```

```
public class Tarea
{
    0 references
    public virtual int Id { get; set; }

    0 references
    public virtual string Descripcion { get; set; }

    0 references
    public virtual IList<Asignacion> Asignaciones { get; set; }
}
```

```
public class Asignacion
{
    0 references
    public virtual int Id { get; set; }

    0 references
    public virtual Usuario Usuario {get; set; }

    0 references
    public virtual Tarea Tarea { get; set; }

    0 references
    public virtual DateTime Desde { get; set; }

    0 references
    public virtual DateTime Hasta { get; set; }
}
```



Mapeo de relaciones entre objetos

Herencia

baufest

Existen tres estrategias soportadas:

Tabla por jerarquía *(table per class hierarchy)*

- Todas las clases de la jerarquía se mapean a la misma tabla.
- La tabla debe tener todos los campos definidos en todas las subclases de la jerarquía. Los campos definidos en las subclases no pueden ser “NOT NULL”.
- Es eficiente e nivel consultas. No es el mejor modelo conceptual

Tabla por subclase *(table per subclass)*

- Cada clase de la jerarquía se mapea a una tabla distinta
- La relación de una clase con su superclase se modela con una relación one-to-one por primary key
- Es el modelo más “correcto”. Hacer una consulta requiere varios joins.

Tabla por clase concreta *(table per concrete class)*

- Cada clase concreta se mapea a una tabla distinta
- Las clases abstractas no se mapean
- Cada tabla tiene que contener los campos de la clase concreta más los de las clases abstractas hacia arriba en la jerarquía
- Reduca la cantidad de joins en consultas. Obliga a repetir campos en las tablas de las clases concretas.

Mapeo de Vistas y Stored Procedures

Vistas

baufest

Mapeo de Vistas

- Se hace de la misma forma que un mapeo a tabla
- La única restricción es que la entidad debería ser declarada inmutable

```
0 references
public class VentaProductoMappingOverride : IAutoMappingOverride<VentaProducto>
{
    1 reference
    public void Override(AutoMapping<VentaProducto> mapping)
    {
        mapping.ReadOnly();
    }
}
```

Mapeo de Vistas y Stored Procedures

Stored Procedures

baufest

En el mapeo de una entidad se puede especificar que las operaciones de insert, update, delete se hagan mediante stored procedures

```
public class ProductoMappingOverride : IAutoMappingOverride<Producto>
{
    2 references
    public void Override(AutoMapping<Producto> mapping)
    {
        mapping.SqlInsert("exec ProductoInsert ?, ?, ?, ?");
        mapping.SqlDelete("exec ProductoDeletet ?");
        mapping.SqlInsert("exec ProductoUpdate ?, ?, ?, ?");
    }
}
```

Práctica

Agregado de objetos relacionados

baufest

- Modificar las convenciones por defecto con el siquiente criterio:
  - Las claves primarias se deben nombrar como Id+NombreClase
  - Las foreign keys se deben nombrar como Id+NombraClaseRelacionada

- Agregar a la API:
  - VentaController con las operaciones de GET y POST (sin PUT/DELETE) para el manejo de las Ventas
  - Tomar modelo de entidades del ejemplo anterior Venta, Item, Cliente, Producto
  - Agregar la opción de Cascade de modo que al guardar la venta se guarden los ítems (override)

baufest

Día 4 – Objetivos del día

- Dudas o consultas de la clase anterior
- Manejo de transacciones
- Mecanismos de consultas
- Revisión del SQL generado por NHibernate mediante un Trace de SQL
- Práctica:
  - Incorporar de transacciones a las operaciones CRUD
  - Consultas con filtros
  - Ejecución de un Trace de SQL

Transacciones

Uso de transacciones

baufest

NHibernate recomienda utilizar transacciones para todas las operaciones

- Usar transacciones asegura la consistencia de los resultados, inclusive en las lecturas

Cuando se modifican objetos es imprescindible el uso de transacciones

- Cuando se modifican objetos, NHibernate puede actualizar la base en cualquier momento con múltiples sentencias
- Hace el tracking de los objetos que se van modificando y puede diferir la ejecución de los comandos a la base de datos para optimizar los accesos.
- Cuando la transacción hace commit, NHibernate toma todos los objetos de la session y los sincroniza con la BD, aunque no se le pida explícitamente que lo haga

Tenemos dos formas de manejar transacciones:

- ITransaction de NHibernate
- TransactionScope de System.Transactions

Transacciones

ITransaction

baufest

- Es el modelo de transacciones propio de NHibernate
- Se implementa utilizando una transacción ADO.Net
- Es un mecanismo más “liviano”
- Es el modelo a utilizar si no se requieren transacciones distribuidas y NHibernate es el encargado de administrar transacciones y conexiones

```
using (var session = sessionfactory.OpenSession())
{
    using (var transacion = session.BeginTransaction())
    {

        var categoriaNotebooks = new Categoria { Nombre = "Notebooks" };
        session.Save(categoriaNotebooks);

        var productoT470 = new Producto
        {
            Nombre = "Lenovo T470",
            Descripcion = "Notebook Lenovo T470 Core i5, 16GB RAM, SSD 128GB",
            Precio = 500,
            Categoria = categoriaNotebooks
        };
        session.Save(productoT470);

        transacion.Commit();
    }
}
```

Transacciones

TransactionScope

baufest

- Permite manejar transacciones en .Net de manera simple mediante el uso de “scopes”
- Es parte del .net framework desde la versión 2.0
- Nhibernate se integra enlistando la conexión a la base en el scope correspondiente para hacer uso de la “ambient transaction”
- El manejo de la transacción se hace por fuera de NHibernate
- Permite mantener una única transacción al hacer operaciones por fuera y dentro de NHibernate
- De ser necesario, se promueve automáticamente a una transacción distribuida administrada por el DTC.

```
using (var scope = new TransactionScope())
{
    using (var session = sessionfactory.OpenSession())
    {
        var categoriaNotebooks = new Categoria { Nombre = "Notebooks" };
        session.Save(categoriaNotebooks);

        var productoT470 = new Producto
        {
            Nombre = "Lenovo T470",
            Descripcion = "Notebook Lenovo T470 Core i5, 16GB RAM, SSD 128GB",
            Precio = 500,
            Categoria = categoriaNotebooks
        };
        session.Save(productoT470);
    }

    scope.Complete();
}
```

Consultas

Mecanismos disponibles

baufest

HQL

Hibernate Query Language es un lenguaje de consultas extremadamente poderoso que se parece bastante a SQL. Sin embargo, no opera sobre tablas y columnas, si no que es orientado a objetos operando sobre objetos y sus propiedades.

Criteria

Es una API orientada a objetos que permite ir componiendo dinámicamente una consultas utilizando distintos objetos y métodos

QueryOver

Se introdujo en la versión 3.0 como un wrapper type-safe alrededor de la API de Criteria, usando extension methods y lambda expressions

LINQ

Implementa un LINQ provider que permite utilizar la API LINQ para escribir consultas con NHibernate

SQL Nativo

También permite el uso de consultas SQL nativas del motor que se esté usando

Consultas

HQL

baufest

Tiene una estructura muy similar al lengaje SQL: select … from … join … where

- Permite hacer asociaciones mediante joins: inner, left, right
- Se usa select cuando se quiere especificar el objeto y propiedades a devolver
- Soporta funciones de agregación avg, sum, min, max, count
- Con la sentencia where se pueden filtrar los resultados. Los operadores son similares a los de SQL
- Permite agrupamientos y ordenamiento con group by y order by
- Se pueden hacer sub-consultas

Una diferencia con SQL es la posibilidad de “navegar” propiedades de los objetos sin necesidad de hacer un join explícito.

Referencia completa: http://nhibernate.info/doc/nhibernate-reference/queryhql.html

Consultas

HQL: Ejemplos

baufest

Ventas a clientes con nombre “Baufest”:

```
var ventasBaufest = session.CreateQuery("from Venta venta where venta.Cliente.Nombre = 'Baufest'").List<Venta>();
```

Ventas con más de un ítem:

```
var ventasMasDeUnItem = session.CreateQuery("from Venta venta where size(venta.Items) > 1").List<Venta>();
```

Productos cuyo Id de Categoría sea 1 ordenados por nombre:

```
var productosNotebook = session.CreateQuery("from Producto prod where prod.Categoria.Id = :idCategoria order by prod.Nombre")
    .SetParameter("idCategoria", 1)
    .List<Producto>();
```

Único Id de producto cuyo nombre contiene ‘T480’

```
var productoId = session.CreateQuery("select prod.Id from Producto prod where prod.Nombre like '%T480%'").UniqueResult<int>();
```

Consultas

HQL: ventajas & desventajas

baufest

Ventajas

- Es muy versátil, tiene un poder similar al lenguaje SQL, por lo que virtualmente cualquier consulta puede escribirse en HQL
- NHibernate cuenta con una cache de consultas HQL pre-compiladas a SQL, por lo que su traducción es muy performante
- Se pueden externalizar las consultas a archivos de configuración, lo que permite que sean pre-compiladas durante la creación de la session factory

Desventajas

- No es type-safe en tiempo de compilación. Un cambio en nombres de clases o propiedades no dará un error de compilación en el HQL hasta el momento de ejecutarlo.
- No es práctico para hacer consultas dinámicas.
- Si bien es similar a SQL requiere el aprendizaje de un nuevo lenguaje

Consultas

Criteria API

baufest

Permite construir consultas dinámicamente utilizando una API orientada a objetos.

- La interface ICriteria representa una consulta sobre una clase persistente
- El contenido del ICriteria se genera componiendo objetos (sigue el patrón composite)
- Se pueden componer consultas de casi cualquier tipo ya que posee la capacitad de generar filtros, proyecciones, ordenamiento, agrupaciones, funciones de agregación.

Consultas

Criteria: Ejemplos

baufest

Ventas a clientes con nombre “Baufest”:

```
var ventasBaufest = session.CreateCriteria<Venta>("venta")
    .CreateAlias("venta.Cliente", "cliente")
    .Add(Restrictions.Eq("cliente.Nombre", "Baufest"))
    .List<Venta>();
```

Productos cuyo Id de Categoría sea 1 ordenados por nombre:

```
var productosNotebook = session.CreateCriteria<Producto>("prod")
    .CreateAlias("prod.Categoria", "cat")
    .Add(Restrictions.Eq("cat.Id", 1))
    .AddOrder(Order.Asc("Nombre"))
    .List<Producto>();
```

Único Id de producto cuyo nombre contiene ‘T480’:

```
var productoId = session.CreateCriteria<Producto>("prod")
    .Add(Restrictions.Like("Nombre", "T480", MatchMode.Anywhere))
    .SetProjection(Projections.Property("Id"))
    .UniqueResult<int>();
```

Consultas

Criteria: ventajas & desventajas

baufest

Ventajas

- Permite construir consultas totalmente dinámicas con una API orientada a objetos (sin concatenar strings)
- Se pueden optimizar las consultas indicando que traiga asociaciones de forma lazy o eager

Desventajas

- La curva de aprendizaje es bastante elevada, requiere un tiempo entender la API
- No es tan intuitivo como el HQL
- Los nombres de las propiedades se ponen como string por lo que no es type safe en tiempo de compilación
- Una consulta escrita con la API ICriteria no es tan fácil de leer e interpretar

Consultas

Query Over API

baufest

Es un wrapper de la API ICriteria que utiliza Extension Methods y Lambda Expressions introducidos en .Net 3.5. La idea era generar una API con el mismo poder que ICriteria pero atacando sus debilidades:

- Se utilizaron Lambda Expressions para reemplazar los magic strings, haciéndola más amigable con los refactorings
- Se utilizaron Extension Methods para hacer la API más “fluent”, más alineado a lo que es LINQ
- Aunque se vea similar a LINQ, **no** es LINQ

Consultas

Query Over: Ejemplos

baufest

Ventas a clientes con nombre “Baufest”:

```
var ventasBaufest = session.QueryOver<Venta>()
    .JoinQueryOver(venta => venta.Cliente)
    .Where(cliente => cliente.Nombre == "Baufest")
    .List();
```

Productos cuyo Id de Categoría sea 1 ordenados por nombre:

```
var productosNotebook = session.QueryOver<Producto>()
    .OrderBy(prod => prod.Nombre).Asc()
    .JoinQueryOver(prod => prod.Categoria)
    .Where(cat => cat.Id == 1)
    .List();
```

Único Id de producto cuyo nombre contiene ‘T480’:

```
var productoId = session.QueryOver<Producto>()
    .WhereRestrictionOn(prod => prod.Nombre).IsLike("T480", MatchMode.Anywhere)
    .Select(prod => prod.Id)
    .SingleOrDefault<int>();
```

Consultas

Query Over: ventajas & desventajas

baufest

Ventajas

- Tiene las ventajas de la API ICriteria
- Es type-safe en tiempo de compilación y refactor friendly
- Es un poco más legible e intuitiva que ICriteria por su sintaxis fluent y lambda expressions más familiar dentro de la programación .Net

Desventajas

- Si bien simplifica ICriteria más que nada para las consultas simples, las consultas complejas todavía son complejas de escribir
- Si bien tiene sintaxis similar a LINQ, tiene otra semántica que genera confusiones acostumbrados a la API de Microsoft

Consultas

LINQ to NHibernate

baufest

Es un provider para LINQ que convierte expresiones en consultas HQL.

Permite escribir consultas utilizando la API estándar de Microsoft que luego podrán ser ejecutadas en NHibernate, que luego las traducirá a SQL.

La implementación esta muy completa y permite convertir la mayoría de las expresiones:

- Joins, proyecciones, funciones de agregación, agrupamientos y ordenamientos
- Funciones de .Net se traducen a SQL: operadores; funciones matemáticas, fechas y strings
- No todo lo que funciona en LINQ to Objects funciona con NHibernate, ya que el provider tiene que ser capaz de traducirlo a SQL
- Las consultas no se ejecutan hasta que no se iteran los resultados o se hace un ToList()

Consultas

LINQ to NHibernate: Ejemplos

baufest

Ventas a clientes con nombre “Baufest”:

```
var ventasBaufest = session.Query<Venta>()
    .Where(venta => venta.Cliente.Nombre == "Baufest")
    .ToList();
```

Ventas con más de un ítem:

```
var ventasMasDeUnItem = session.Query<Venta>()
    .Where(venta => venta.Items.Count > 1)
    .ToList();
```

Productos cuyo Id de Categoría sea 1 ordenados por nombre:

```
var idCategoria = 1;
var productosNotebook = session.Query<Producto>()
    .Where(prod => prod.Categoria.Id == idCategoria)
    .OrderBy(prod => prod.Nombre);
```

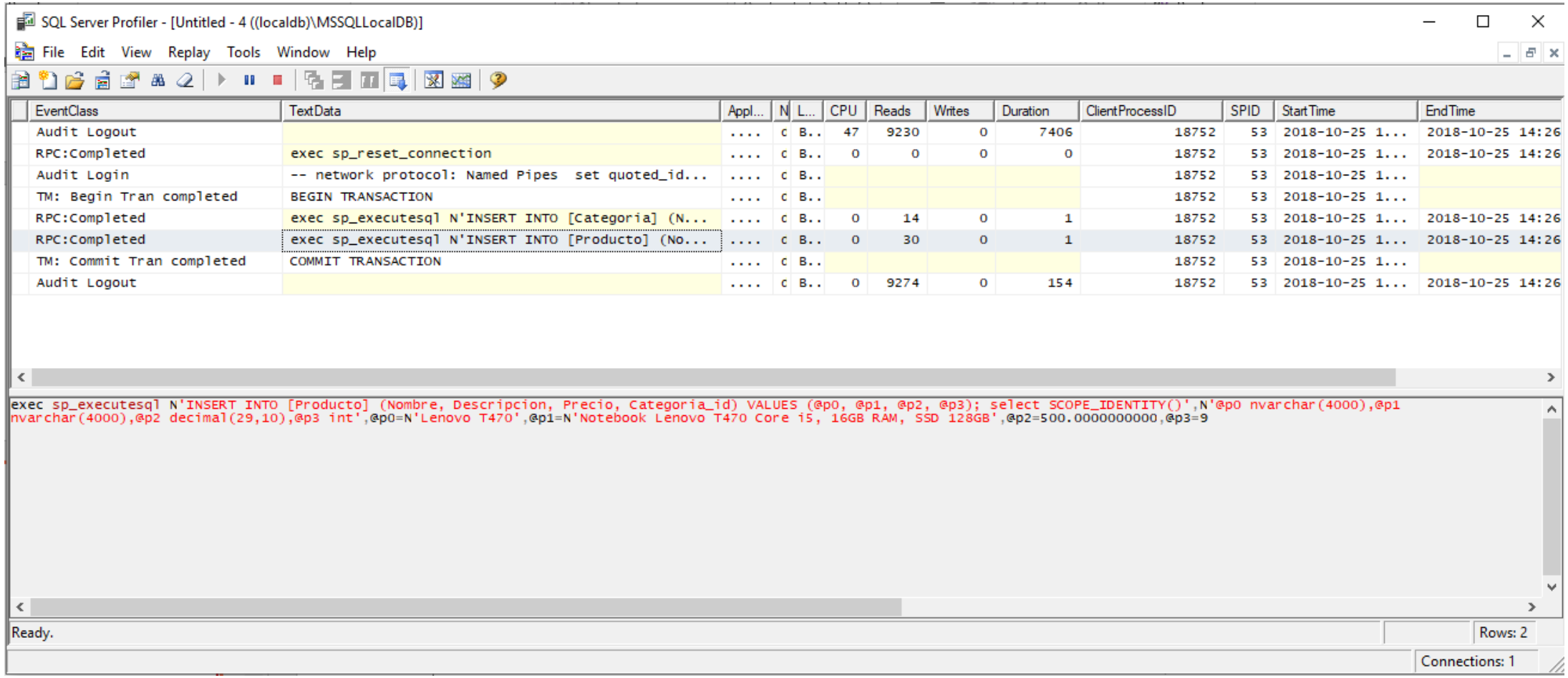
Único Id de producto cuyo nombre contiene ‘T480’:

```
var productoId = session.Query<Producto>()
    .Where(prod => prod.Nombre.Contains("T480"))
    .Select(prod => prod.Id)
    .SingleOrDefault();
```

Revisión de SQL

SQL Server Profiler

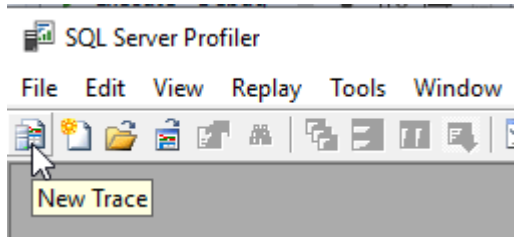
baufest

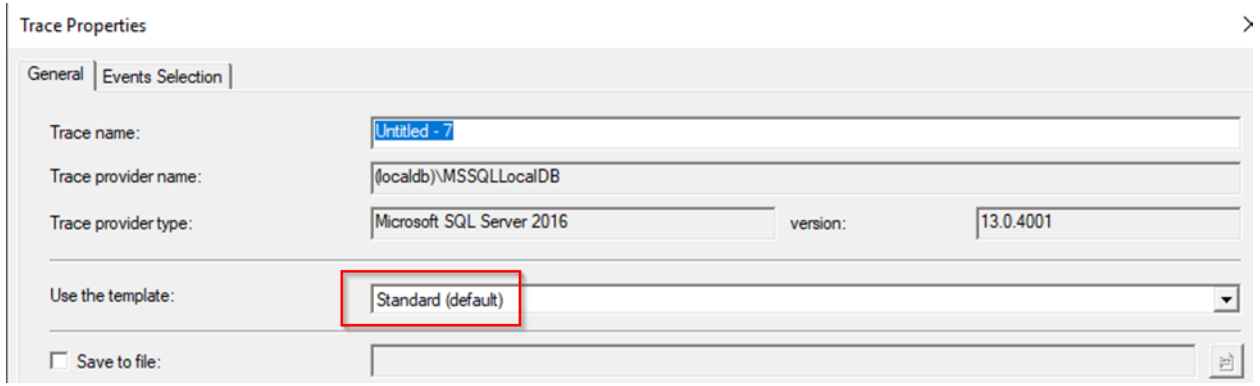


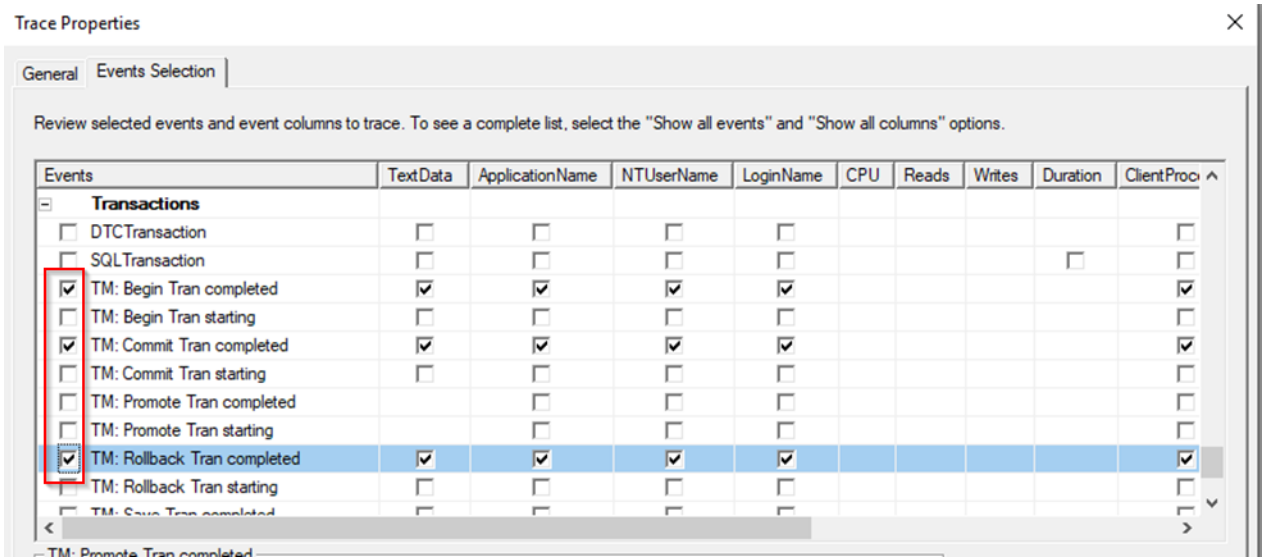
Revisión de SQL

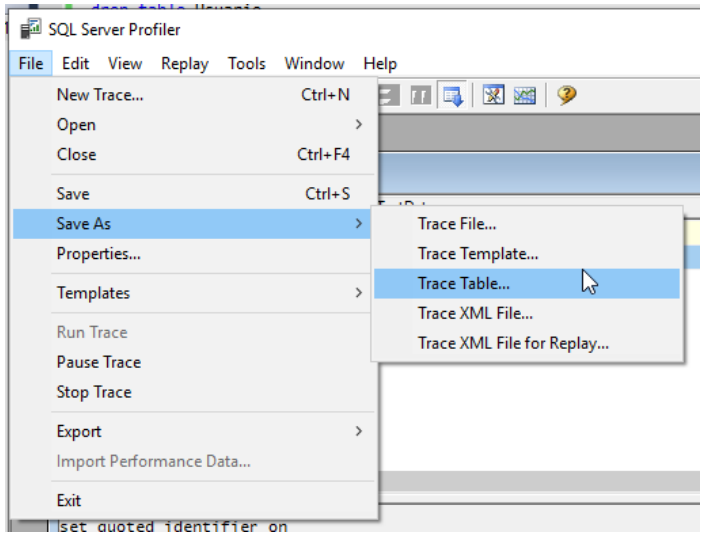
SQL Server Profiler: crear una traza

baufest









Práctica

Transacciones y Consultas

baufest

- Agregar transacciones a los controllers de Producto, Categoria y Venta en los métodos Post, Put y Delete
  - En Producto usar TransactionScope
  - En Categoria y Venta usar Itransaction

- Agregar un filtro por nombre de categoría (opcional) en el métotodo Get de CategoriaController que lista categorías:

```
public IEnumerable<Categoria> Get([FromUri]string nombre)
{
    using(var session = Database.SessionFactory.OpenSession())
    {
        var categorias = session.Query<Categoria>();
        if (!string.IsNullOrEmpty(nombre))
        {
            categorias = categorias.Where(cat => cat.Nombre.Contains(nombre));
        }

        return categorias.ToList();
    }
}
```

Práctica

Transacciones y Consultas

baufest

- Agregar un filtro por nombre, descripción e IdCategoria (todos opcionales) en el método Get de ProductoController que lista productos:

```
public IEnumerable<ProductoDto> Get([FromUri]FiltroProducto filtro)
{
```

```
namespace Baufest.NHibernate.Api.Models
{
    1 reference | 0 changes | 0 authors, 0 changes
    public class FiltroProducto
    {
        2 references | 0 changes | 0 authors, 0 changes
        public string Nombre { get; set; }

        2 references | 0 changes | 0 authors, 0 changes
        public string Descripcion { get; set; }

        2 references | 0 changes | 0 authors, 0 changes
        public int? IdCategoria { get; set; }

    }
}
```

Práctica

Transacciones y Consultas

baufest

- Agregar un filtro por IdCliente e IdProducto (ambos opcionales) en el método Get de VentaController que lista Ventas:
  - Si se especifica IdCliente solo lista ventas de ese cliente
  - Si se especifica IdProducto solo lista ventas que contengan ese producto

```
public IEnumerable<VentaDto> Get([FromUri]FiltroVenta filtro)
{
```

```
public class FiltroVenta
{
    2 references | 0 changes | 0 authors, 0 changes
    public int? IdCliente { get; set; }

    2 references | 0 changes | 0 authors, 0 changes
    public int? IdProducto { get; set; }
}
```

```
ventas.Where(venta => venta.Items.Any(item => item.Producto.Id == filtro.IdProducto));
```

baufest

Día 5 – Objetivos del día

- Dudas o consultas de la clase anterior
- SQL Nativo
- Buenas y malas prácticas
- Uso en aplicaciones web
- Práctica:
  - Consultas con SQL nativo
  - Implementar algunos de los tips

Consultas

SQL Nativo

baufest

NHibernate permite escribir consultas en el dialecto SQL nativo del motor de base de datos.

Es útil principalmente cuando:

Se necesitan usar características o funciones específicas el motor

Hacer optimizaciones puntuales en consultas o procedimientos que con el ORM no se pueden lograr

Pueden utilizarse cualquiera de los métodos que provea el motor:

- Consultas o sentencias SQL de cualquier tipo
- Stored procedures
- Funciones

“Un gran poder conlleva una gran responsabilidad”

Consultas

SQL Nativo: Ejemplos

baufest

Ventas a clientes con nombre “Baufest”:

```
var ventasBaufest = session.CreateSQLQuery(
    @"SELECT venta.Id, venta.Cliente_id
        FROM Venta venta
        INNER JOIN Cliente cliente ON venta.Cliente_id = cliente.Id
        WHERE cliente.Nombre = 'Baufest'").AddEntity(typeof(Venta)).List<Venta>();
```

Ventas con más de un ítem:

```
var ventasMasDeUnItem = session.CreateSQLQuery(
    @"SELECT venta.Id, venta.Cliente_id
        FROM Venta venta
        WHERE (SELECT COUNT(item.Id) FROM Item item WHERE item.Venta_id = venta.Id) > 1")
    .AddEntity(typeof(Venta)).List<Venta>();
```

Productos cuyo Id de Categoría sea 1 ordenados por nombre:

```
var productosNotebook = session.CreateSQLQuery(
    @"SELECT prod.Id, prod.Nombre, prod.Descripcion, prod.Precio, prod.Categoria_id
        FROM Producto prod
        WHERE prod.Categoria_id = :idCategoria
        ORDER BY prod.Nombre").AddEntity(typeof(Producto)).SetParameter("idCategoria", 1).List<Producto>();
```

Único Id de producto cuyo nombre contiene ‘T480’

```
var productoId = session.CreateQuery(
    @"SELECT prod.Id
        FROM Producto prod
        WHERE prod.Nombre LIKE '%T480%'").UniqueResult<int>();
```

Consultas

SQL Nativo: ventajas & desventajas

baufest

Ventajas

- Permite ejecutar SQL con todas las funciones del motor que se está utilizando
- Es útil para implementar del lado de la base de datos procesos de movimiento masivo de datos que son poco performantes usando un ORM
- El manejo de las conexiones y la transacción se siguen haciendo con NHibernate
- Permite hacer optimizaciones en consultas o procesos puntuales
- Se puede mapear el resultado de una consulta SQL a una entidad

Desventajas

- Genera acoplamiento con el motor de base de datos al “saltear” la capa del ORM
- Modificaciones en la base de datos implica modificación de las consultas
- Si se abusa de este tipo se van perdiendo los beneficios del uso de NHibernate

Tips de Performance

No siempre hace falta un stored procedure

baufest

- Al usar un ORM, por olvidar que hay una base de datos detrás, se pueden cometer errores que afecten negativamente la performance de la aplicación.
- En muchos casos no es necesario recurrir a SQL nativo o Stored Procedures
- Cambios sutiles en las consultas LINQ pueden generar grandes cambios en la performance
- Siempre conviene tener a mano un profiler para revisar las consultas que se están generando

Tips de Performance

Lazy Loading vs Select new

baufest

- Armar la consulta para generar un objeto, únicamente con las propiedades de cada entidad que vamos a utilizar.
  - No tiene sentido levantar 20 columnas de una tabla si solo necesitamos 3 de ellas para la pieza de código que se está ejecutando.
  - Con esta técnica, además de reducir la cantidad de datos que leemos de la base de datos, evitamos generar una gran cantidad de desencadenadas por el lazy loading (conocido como el problema de select n+1).

```
12 references
public class Product
{
    2 references
    public virtual int Id { get; set; }
    4 references
    public virtual string Code { get; set; }
    5 references
    public virtual string Description { get; set; }
    3 references
    public virtual decimal RetailPrice { get; set; }
    6 references
    public virtual Category Category { get; set; }
}
```

```
var products = session.Query<Product>().Where(x => x.Description.Contains("Bike"));

var model = new List<ProductModel>();
foreach (var product in products)
{
    model.Add(new ProductModel
    {
        Id = product.Id,
        Description = product.Description,
        CategoryId = product.Category.Id,
        CategoryName = product.Category.Description,
    });
}
```

Tips de Performance

Lazy Loading vs Select new

baufest

- Armar la consulta para generar un objeto, únicamente con las propiedades de cada entidad que vamos a utilizar.
  - No tiene sentido levantar 20 columnas de una tabla si solo necesitamos 3 de ellas para la pieza de código que se está ejecutando.
  - Con esta técnica, además de reducir la cantidad de datos que leemos de la base de datos, evitamos generar una gran cantidad de desencadenadas por el lazy loading (conocido como el problema de select n+1).

```
12 references
public class Product
{
    2 references
    public virtual int Id { get; set; }
    4 references
    public virtual string Code { get; set; }
    5 references
    public virtual string Description { get; set; }
    3 references
    public virtual decimal RetailPrice { get; set; }
    6 references
    public virtual Category Category { get; set; }
}
```

```
var model = session.Query<Product>()
    .Where(x => x.Description.Contains("Bike"))
    .Select(x => new ProductModel
    {
        Id = x.Id,
        Description = x.Description,
        CategoryId = x.Category.Id,
        CategoryName = x.Category.Description,
    });
```

Tips de Performance

Fetch

baufest

- Se puede indicar a NH que además de la entidad obtenga una o más de sus relaciones, evitando el lazy loading el método *Fetch*
  - Se debe indicar por cada relación que se quiera obtener de forma *Eager*
  - Tambien existe el método FetchMany que se utiliza cuando la relación es una colección
  - Si se desean obtener más de un nivel existe el método *ThenFetch.*

```
var productosNotebook = session.Query<Producto>()
    .Fetch(prod => prod.Categoria)
    .Where(prod => prod.Categoria.Id == idCategoria)
    .OrderBy(prod => prod.Nombre)
    .ToList();
```

```
var items = session.Query<Item>()
   .Fetch(item => item.Producto).ThenFetch(prod => prod.Categoria)
   .Where(item => item.Cantidad > 1)
   .ToList();
```

Tips de Performance

Count vs Any

baufest

- Para conocer la existencia de una entidad o un registro bajo ciertos criterios, siempre conviene usar **“Any”** en lugar de un **“Count”**.
  - Any será traducido a un Exists o a un Select TOP, dependiendo del ORM
  - Count será traducido a la función de agregación de SQL count que tiene más costo de ejecución.

```
var existe = session.Query<Producto>()
    .Count(prod => prod.Nombre.Contains("T480")) > 0;
```

```
var existe = session.Query<Producto>()
      .Any(prod => prod.Nombre.Contains("T480"));
```

Tips de Performance

ToList()

baufest

- Una consulta mediante LINQ no se ejecuta contra la base de datos hasta que no se enumeran los resultados.
  - Esta enumeración puede ocurrir al hacer un foreach sobre los resultados, o al hacer un ToList.
  - Se deben aplicar los filtros sobre la tabla antes de hacer el ToList.

```
var products = session.Query<Product>().ToList();
var filteredProducts = products.Where(x => x.Description.Contains("Bike")).ToList();
```

```
var products = session.Query<Product>();
var filteredProducts = products.Where(x => x.Description.Contains("Bike")).ToList();
```

Tips de Performance

Relaciones bidireccionales

baufest

- Muchas veces, cuando se mapean objetos relacionados, la relación se arma de forma bidireccional.
- Cuando agregamos una propiedad con una lista o colección, hay que tener certeza de que serán cantidades razonables de objetos relacionados y no crecerán con el tiempo

```
2 references
public class ItemDeVenta
{
    0 references
    public int Id { get; set; }
    0 references
    public Venta Venta { get; set; }
    0 references
    public Producto Producto { get; set; }
}
```

```
1 reference
public class Producto
{
    0 references
    public int Id { get; set; }
    0 references
    public string Nombre { get; set; }
    0 references
    public IList<ItemDeVenta> ItemsDeVenta { get; set; }
}
```

```
1 reference
public class Venta
{
    0 references
    int Id { get; set; }
    0 references
    IList<ItemDeVenta> Items { get; set; }
}
```

Tips de Performance

Relaciones bidireccionales (2)

baufest

- Una sentencia como la siguiente, puede desatar que el ORM levante millones de registros de ItemDeVenta:

```
var items = producto.ItemsDeVenta;
```

- Incluso sin acceder directamente a la propiedad, el ORM puede accederla para mantener la consistencia de la relación cuando accedemos la propiedad opuesta:

```
itemDeVenta.Producto = producto;
```

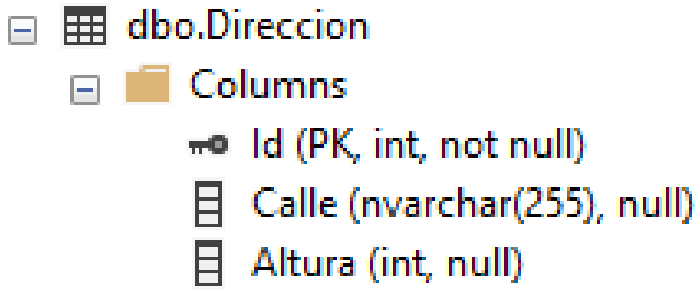
El ORM puede internamente acceder a la propiedad ItemsDeVenta de la clase Producto para mantener la consistencia de la relación bidireccional, por lo que, potencialmente estaríamos obteniendo de la base de datos, gran cantidad de registros sin siquiera acceder a una propiedad que es una colección. Acá es cuando el DBA te dice que en su profiler ve que estás haciendo una consulta que levanta un millón de registros y vos le juras que no estás haciendo ninguna consulta así.

Tips de Performance

Nullable value types

baufest

- Es importante tener cuidado al usar Value Types.
  - Si el tipo es nulleable en la tabla, entonces hay que mantenerlo también a nivel de objetos.
- Cuando se presenta esta inconsistencia entre modelo de tablas y objetos, los ORM suelen detectar que la entidad tuvo un cambio y tratan de actualizar el campo en la tabla con el valor default del Value Type.




```
public class Direccion
{
    0 references | 0 changes | 0 authors, 0 changes
    public virtual int Id { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public virtual string Calle { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public virtual int Altura { get; set; }
}
```

```
public class Direccion
{
    0 references | 0 changes | 0 authors, 0 changes
    public virtual int Id { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public virtual string Calle { get; set; }

    0 references | 0 changes | 0 authors, 0 changes
    public virtual int? Altura { get; set; }
}
```

Tips de Performance

Conclusión

baufest

- Estos son solo algunos tips puntuales. Siempre se recomienda verificar qué consulta SQL se está generando a partir del ORM.

- Cada ORM tiene su particularidad a la hora de traducir el código a una consulta:
  - No se debe dejar de controlar que las consultas generadas sean performantes
  - Investigar si hay alguna mejor forma de escribir el código LINQ para generar un mejor SQL.

- En la mayoría de los casos se pueden lograr consultas eficientes y performantes sin necesidad de tener que escribir la consulta directamente en SQL o programar un Stored Procedure.

- Es fundamental no desentenderse de la base de datos
  - Adoptar la práctica de verificar como terminan traduciéndose a SQL las consultas que escribimos con LINQ, particularmente cuando son complejas.
  - Es imprescindible conocer los detalles y limitaciones que posee el ORM que estamos utilizando y las características del motor de base de datos sobre el cual estamos persistiendo nuestros datos.

Uso en aplicaciones Web

Session per request

baufest

- Para manejar la sesión de Nhibernate en aplicaciones Web se suele crear una sesión por cada Request, de manera que acompañe el ciclo de vida de la aplicación.
- La ventaja de este patrón es que encapsula todas las operaciones de un request en una unidad de trabajo, y lo mantiene independiente de otros request concurrentes.
- Hay varias puntos donde se puede crear o liberar la sesión, por ejemplo en Global.asax o ActionFilters.
- La manera recomendada de manejar la sesión es usar un framework de inyección de dependencias que se ocupe de este tema.

Uso en aplicaciones Web

Inyección de dependencias

baufest

- Se pueden configurar los frameworks de inyección de dependencia para manejar la sesión de NHibernate por request.
- Se deben configurar dos mapeos: ISessionFactory e ISession
- En el ejemplo usamos NInject

```
kernel.Bind<ISessionFactory>()
    .ToMethod(context => Database.CreateSessionFactory())
    .InSingletonScope();

kernel.Bind<ISession>()
    .ToMethod(context => context.Kernel.Get<ISessionFactory>().OpenSession())
    .InRequestScope();
```

```
public class CategoriaController : ApiController
{
    private ISession session;

    0 references | 0 changes | 0 authors, 0 changes
    public CategoriaController(ISession session)
    {
        this.session = session;
    }

    // GET: api/Categoria
    [Description("Lista todas las categorias disponibles")]
    0 references | 0 changes | 0 authors, 0 changes
    public IEnumerable<Categoria> Get([FromUri]string nombre = null)
    {
        var categorias = session.Query<Categoria>();
        if (!string.IsNullOrEmpty(nombre))
        {
            categorias = categorias.Where(cat => cat.Nombre.Contains(nombre));
        }

        return categorias.ToList();
    }
```

Links y Referencias

baufest

NHibernate

- Web oficial: http://nhibernate.info/
- Documentación: http://nhibernate.info/doc/index.html
- Repo GitHub: https://github.com/nhibernate/nhibernate-core

FluentNHibernate

Web oficial: https://www.fluentnhibernate.org/

Documentación: https://github.com/FluentNHibernate/fluent-nhibernate/wiki/Getting-started

Repo GitHub: https://github.com/FluentNHibernate/fluent-nhibernate

Práctica

SQL Nativo & Optimización

baufest

SQL Nativo:

- Reescribir la consulta del método Get de CategoriaController que filtra por nombre en SQL Nativo.
- Reescribir la consulta del método Get de ProductoController que filtra por nombre y descripción en SQL Nativo.

Optimización (select-new):

- Experimentar con el método Get de VentaController que filtra por cliente y producto
  - Ejecutar el proyecto de ejemplo y verificar en el profiler que consultas genera el método
  - Agregar un ToList() antes de convertir a DTO y revisar nuevamente el SQL ejecutado

```
var ventasListadas = ventas.ToList();

return ventasListadas.Select(venta => new VentaDto
{
    Id = venta.Id,
    IdCliente = venta.Cliente.Id,
    Items = venta.Items.Select(i => new ItemDto
                                    {
                                        IdProducto = i.Producto.Id,
                                        NombreProducto = i.Producto.Nombre,
                                        Cantidad = i.Cantidad
                                    }).ToList()
}).ToList();
```

Práctica

SQL Nativo & Optimización

baufest

Optimización (Fetch):

- Experimentar con el método Get de VentaController que filtra por cliente y producto
  - Mantener el ToList agregado en el punto anterior
  - Usar los mpetodos FetchMany y ThenFetch para cargar de forma eager las asociaciones de ítems y productos
  - Verificarlos con el profiler

```
ventas = ventas.FetchMany(vta => vta.Items).ThenFetch(it => it.Producto);
```

Día 6 – Objetivos del día

- Ejercicio integrador
- Dudas o consultas del curso
- Despedida

Ejercicio Integrador

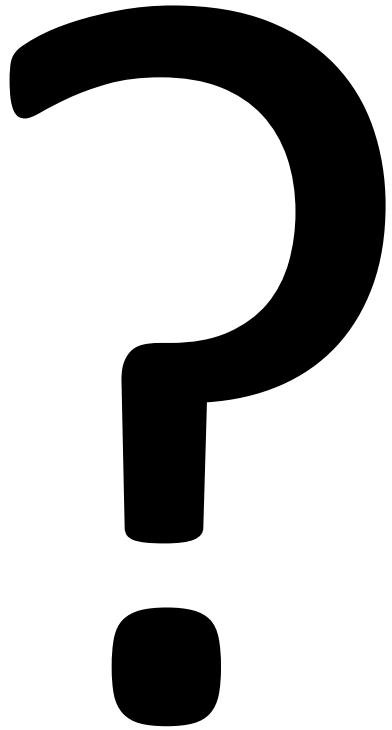
Preguntas & Respuestas

baufest

Consultas

Preguntas / Dudas / Sugerencias

baufest



baufest

Muchas gracias!