NHibernate course

# Intro

Wat niet?

* Basis ORM concepten
* Inleiding tot NHibernate
* Hoe entiteiten naar DB mappen? Mapping files, Fluent NH, ConfORM? I don’t care.
* Hoe queryen? Linq, QueryOver, HQL? Don’t care either.

Wat dan wel?

* Veelvoorkomende problemen oplossen
* Tips & tricks / best practices
* Designing NH applications
* Extending NH

# Veelvoorkomende problemen

## De SELECT N+1

Veruit het meest voorkomende probleem. Wat is het juist? *(Show the code: demo 1)*

* 1 query om de root entity op te halen
* N queries om child entities op te halen
* Enorme performance loss

Hoe lossen we dit op?

* Eager loading:
  + fetch = join in mapping
  + left.joinqueryover in query

Zet altijd een profiler op wanneer je een ORM gebruikt!

## Duplicate records

Is een gevolg van eager loading m.b.v een join.

Aanpassingen aan data model: een book kan geschreven zijn door meerdere authors.  
Indien we een lijst van books ophalen samen met hun author zal elk book N keer in de resultset staan, waarbij N het aantal authors is waardoor het book geschreven is. *(Show the code: demo 2)*

Hoe lossen we dit op?

* Oplossing 1: DistinctRootEntity transformer (slecht als je veel records of joins hebt)
* Oplossing 2: Future queries
* Oplossing 3: fetch = subselect in mapping

# Tips & tricks / best practices

## Primary key strategies

* Zelf de PK assignen: assigned
* Database de PK laten assignen: identity, sequence, native
* NH de PK laten assignen: guid, guid.comb, hilo, …

Vermijd database assigned keys. Doordat de key door de DB assigned is moet NH een SELECT doen om de waarde ervan op te halen. Dit is slechts voor de performance en zegt ervoor dat de inserts niets gebatched kunnen worden.

Wat is een primary key:

* PKs zijn meestal de clustered index van tables. De records van een table worden opgeslagen volgens de volgorde van de column waarop de clustered index gedefinieerd is. Dus deze waardes die we in deze column steken moeten moeten zo sequencieel mogelijk zijn zodat we index fragmentation kunnen beperken.

Beste keuze is guid.comb & hilo. Waarom?

* De guid.comb strategy genereert een sequenciële guid
* De hilo strategy reserveert ranges van sequenciële integers per session factory

Geef voorbeeld van guid.comb & hilo *(Show the code: demo 3)*

## Component mapping

Toon Customer table in DB. Een customer heeft een BillingAddress, ShippingAddress & LegalAddress. Elk adres bestaat uit Street, City & Country. In onze model willen we de adres informatie mappen naar een Address component. Toon Customer table versus Customer entity en de mapping die ervoor nodig is. *(Show the code: demo 4)*

## Various

Mapping intellisense: hoe krijg ik intellisense in mijn mapping files en config file? Voeg de nhibernate XSDs toe aan je solution.

Dynamic update: update enkel de dirty fields. *(Show the code: demo 5)*

* Veel performanter voor de database
* Veel minder onnodige gegevens in de transaction log
* Nadeel: voor elk nieuw statement wordt een execution plan aangemaakt dat mogelijk nooit meer wordt uitgevoerd en dus onnodige ruimte in beslag neemt. Dit kunnen we m.b.v. een SQL job onder controle houden.

NHibernate logging: NHibernate kan geïntegreerd worden met log4net. Toon configuratie en demo. *(Show the code: demo 5). De NHibernate profiler werkt door in te haken op de informatie wordt weggeschreven door NHibernate naar de log4net log.*

# Designing NH application

Abstracting an ORM with the Repository pattern:

*I would like to design an application using NHibernate. But I also want to so flexible that in future ,if I unplug NHibernate and use Entity framework or another framework then my application should not crash.(Show the code: demo 6)*

Dit werkt niet.

* Er zijn verschillende in gedrag tussen verschillende ORM die je niet kan abstraheren. Bijvoorbeeld: NH gaat anders om met batching, transaction handling & caching dan EF.
* Kost veel tijd om bepaalde features te abstraheren.
* Je kan enkel features abstraheren die de ORMs gemeenschappelijk hebben. Dit heeft als gevolg dat je geen gebruik kan maken van advanced features die bepaalde ORMs aanbieden.
* Keuze van de ORM kan ook invloed hebben op andere zaken dan data access. Zo moeten alle public properties/method voor NHibernate virtual zijn om lazy loading mogelijk te maken.

Dus we moeten ons niets meer aantrekken van encapsulation? Nee, by all means encapsuleer uw data access code, maar abstraheer niet de API van uw ORM.

Hoe doe ik dit dan? Wat is dan een goeie NHibernate architectuur? *(Show the code: demo 7)*

* Entities zijn objecten een datastructures: encapsulate data, expose behaviour
* Encapsuleer queries
* Don’t return entities to the UI
* Centraal beheer van transacties: out of scope voor demo

Transaction best practices:

* TX per web request
* ASP.NET: http module
* ASP.NET MVC: BaseController, ActionFilter

# Extending NH

## EventListeners

Implement auditing *(Show the code: demo 7)*

## Interceptors

Show don’t hurt me interceptor *(Show the code: demo 7)*