

Paweł Rajba

pawel@cs.uni.wroc.pl

<http://pawel.ii.uni.wroc.pl/>

NHibernate

Agenda

- Wprowadzenie
- Architektura
- Trwałość przezroczysta
- Konfiguracja, konfiguracja mapowania
- Dziedziczenie klas
- Kolekcje
- Asocjacje
- Cykl życia obiektów
- Trwałość przechodnia
- Strategie sprowadzania danych
- Pobieranie obiektów
- Cache system
- Stateless session
- Debugging

Utrwalanie obiektów

- Pojęcie utrwalania danych
 - Składowanie danych przetwarzanych podczas działania programu
 - Najczęściej w bazie relacyjnej, które są obecnie (wciąż) najbardziej rozpowszechnione
- Obiekty trwałe i ulotne
- Trwałość przechodnia, czyli trwałość przez osiągalność
- Trwałość będziemy rozumieć jako połączenie elementów
 - zapamiętywanie, organizacja i pobieranie danych,
 - współbieżność i integralność danych,
 - współdzielenie danych.

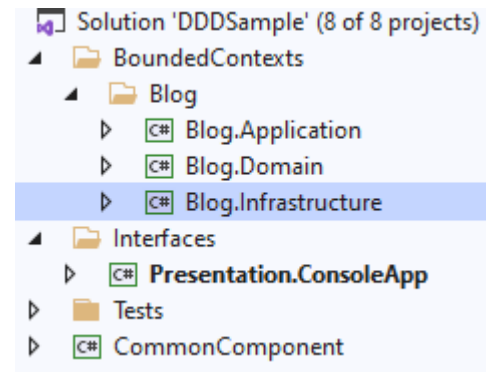
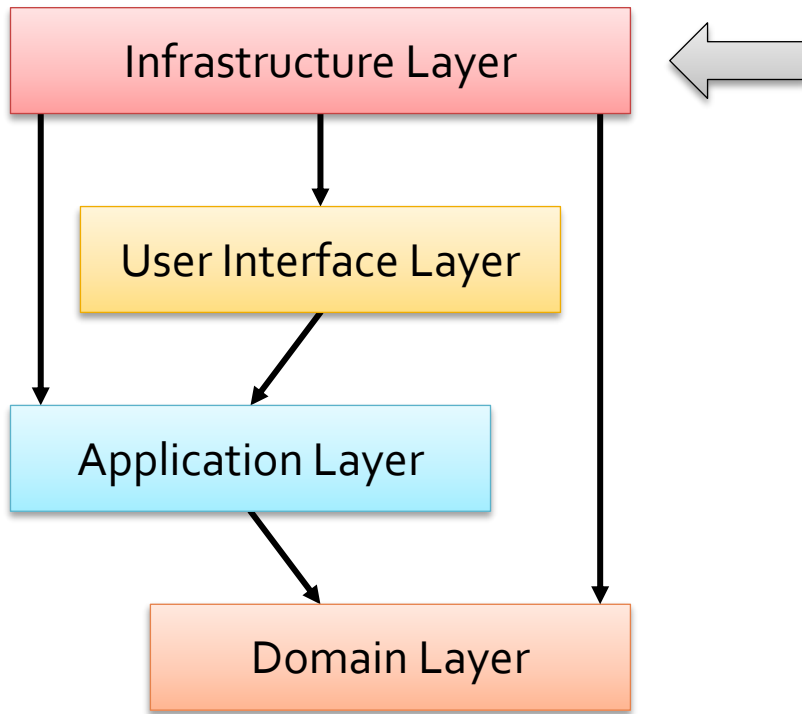
Utrwalanie obiektów

- Jak można inaczej?
 - Serializacja
 - Prawie każda technologią wspiera ten mechanizm
 - Zwykle w wyniku dostajemy pliki XML lub JSON
 - Jednak serializacja nie umożliwia:
 - Formułowania zapytań
 - Częściowego odczytu bądź aktualizacji
 - Zarządzania cyklem życia obiektów
 - Współbieżności i transakcji
 - Niemniej serializację będziemy wykorzystywać do przygotowania obiektów do przesłania przez sieć

Utrwalanie obiektów

- Jak można inaczej?
 - „Prosty schemat”
 - generujemy przygotowane zapytania dla obiektów
 - parametryzujemy zapytania dla konkretnych obiektów, przeglądamy wyniki zapytań,...
 - Taki schemat realizuje dostęp niskopoziomowy
 - Zadanie dosyć żmudne
 - Programista powinien się skupić na tworzeniu logiki biznesowej, a nie obsłudze trwałości

Gdzie jesteśmy?



Inne rozwiązania

- Mapowanie do MongoDB
- Mapowanie do Neo4j

Problemy niedopasowania

- Polega na zupełnie innych filozofiach dotyczących modelu relacyjnego i obiektowego
 - tzw. niedopasowanie paradygmatów
- Niedopasowanie implikuje szereg konkretnych problemów. Przykładowe:
 - Problem szczegółowości
 - Problem podtypów
 - Problem identyczności
 - Problem asocjacji
 - Problemy nawigacji po grafie obiektów

Problemy niedopasowania

- Problem szczegółowości
 - Mamy klasy User i Address
 - W bazie danych możemy utworzyć dla nich osobne tabele
 - pojawia się problem dużych złączeń
 - Albo zapamiętać poszczególne pola adresu w tabeli User: User(ID, Name, A_Street, A_City, A_Code)
 - i wtedy pojawia się problem szczegółowości.
 - Problem łatwy do rozwiązania, chociaż często spotykany

Problemy niedopasowania

- Problem podtypów
 - W większości DBMS nie jest obsługiwane dziedziczenie
 - Z drugiej strony dziedziczenie to podstawowy mechanizm w językach obiektowych
 - Zagadnienie asocjacji polimorficznej. Rozpatrzmy przykład:
 - Mamy klasy User —^{1..*}— Payment, CreditCard → Payment, BankAccount → Payment
 - Powiązanie User–Payment realizuje asocjację polimorficzną

Problemy niedopasowania

- Problem identyczności
 - Jak możemy porównywać elementy:
 - Za pomocą porównania obiektów operatorem ==
 - Za pomocą zdefiniowania metody Equals()
 - Porównując klucz główny w tabeli relacyjnej
 - Oczywiście, wszystkie te sposoby się istotnie różnią
 - Pojawia się problem występowania wielu obiektów reprezentujących ten sam wiersz z tabeli relacyjnej
 - Pierwsze zalecenie: jako klucz główny powinno być pole niezależne od innych, będące int-em

Problemy niedopasowania

- Problem asocjacji
 - W systemie relacyjnym mamy związki jeden-do-wielu i jeden-do-jednego
 - Związek wiele-do-wielu jest tak naprawdę połączeniem związków jeden-do-wielu
 - W świecie obiektowym poprzez kompozycję możemy z kolei
 - tworzyć asocjacje jednokierunkową
 - tworzyć asocjacje dwukierunkową, definiując odpowiednie elementy w obu obiektach
 - tworzyć asocjacje jeden-do-jednego, jeden-do-wielu, wiele-do-wielu
 - Ponieważ w obiektowości można więcej, w systemie relacyjnym trzeba dokonywać symulacji

Problemy niedopasowania

- Problemy nawigacji po grafie obiektów
 - Poprzez utworzenie odpowiednich asocjacji, łatwiej nawigować po grafie obiektów
 - Mamy daną ścieżkę: `jednostka.getOsoba(3).getUlica()`
Jak to pobrać?
 - Efektywnie byłoby wykonać odpowiedni JOIN
 - Metody jednak będą ściągać dane po trochu, czyli bardzo nieefektywnie
 - Ogólnie pojawia się problem odwzorowań języka wewnętrznego systemu ORM na odpowiedni dialekt SQL

Systemy ORM

- Czym są?
 - Rozwiązują większość problemów wynikających z niedopasowania paradygmatów
 - Translacja działa na podstawie metadanych opisujących odwzorowanie obiektu na dane
 - Translacja jest przezroczysta dla programisty i działa w obie strony

Systemy ORM

- Główne składowe ORM
 - Interfejs pozwalający na wykonywanie operacji CRUD na obiektach klas umiejących zapewnić trwałość
 - „Język” pozwalający zadawać zapytania w modelu obiektowym
 - Sposób określania metadanych
 - Często wspierane przez narzędzia lub konwencje
 - Elementy dodatkowe: obsługa transakcji, leniwe pobieranie asocjacji, optymalizacje

Systemy ORM

- Pytania dot. systemów ORM
 - Jak musi wyglądać klasa, żeby można było ją utrwalać?
 - Jak definiuje się metadane? Czy są narzędzia, które robią to automatycznie? Czy trzeba je w ogóle definiować?
 - W jaki sposób jest odwzorowywana hierarchia dziedziczenia?
 - Jak realizowane są zagadnienia:
 - Atrybut „not null”,
 - Dostępność pól: public, private, protected,
 - Nazewnictwo
 - np. w Oracle nazwy mogą mieć co najwyżej 30 znaków

Systemy ORM

- Pytania dot. systemów ORM (c.d.)
 - Jak realizowana jest tożsamość obiektów?
 - Jak wygląda współpraca pomiędzy obiektami logiki biznesowej a obiektami oprogramowania ORM?
 - Jakie są możliwości języka zapytań?
 - Jak wydajne jest pobieranie danych z asocjacji?
 - Nasz przykład: `jednostka.getOsoba(3).getUlica()`
 - Ogólniej: jakie są strategie pobierania danych?

Systemy ORM

- Zalety korzystania z ORM
 - Produktywność
 - programista skupia się na problemie biznesowym, a nie składowaniem obiektów
 - Utrzymanie
 - Mniej kodu przez co łatwiej panować nad aplikacją
 - ORM jest zwykle bardziej elastyczny niż własna warstwa dostępu do danych, przez co łatwiej modyfikować aplikację
 - Wydajność
 - Powszechnie panuje przekonanie, że ręcznie napisana trwałość będzie wydajniejsza od tej zautomatyzowanej w ORM
 - I zwykle tak jest, jednak dobry ORM jest dobrze zoptymalizowany i narzut jest niewielki
 - Najczęściej problemy z wydajnością nie wynikają z ORM-a
 - Niezależność od dostawcy
 - Jedna z istotniejszych zalet: zwykle raz napisana aplikacja będzie działać na Oracle, SQL Server, PostgreSQL, itd. ... chociaż to nie do końca prawda.

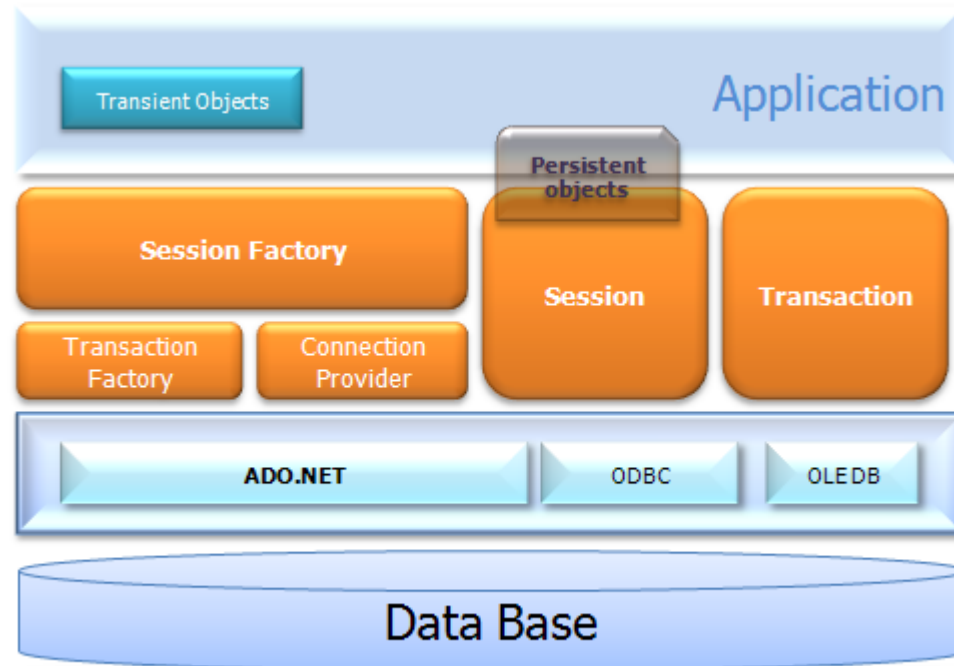
Systemy ORM

- Prawie każda technologia obiektowa dysponuje zwykle co najmniej jednym systemem ORM
- Nierzadko również dana technologia ma ORM wbudowany w framework
 - Np. Django, RoR, Symfony
- W dalszej części omówimy NHibernate dla .NET
 - Jest to port biblioteki Hibernate z Java

NH zaczniemy...

- ...od przykładu
 - HelloWorld

Architektura



Architektura

- Podstawowe interfejsy
 - ISession
 - ISessionFactory
 - ISession vs. ISessionFactory
 - Configuration
 - Transaction
 - IQuery
 - ICriteria

Trwałość przezroczysta

- Polega na tym, że utrwalane klasy nie są świadome utrwalania
 - Kontrwzorzec: ActiveRecord
- NHibernate jest (prawie) przezroczysty
 - Nie trzeba nic implementować ani nic dziedziczyć
 - Za utrwalanie odpowiada menedżer trwałości: interfejsy Session i Query
 - Gdzie jest to prawie?
 - Pola muszą być virtual
 - Czasami typy muszą pochodzić z np. Iesi.Collection

Konfiguracja

- Mamy dwa główne typy konfiguracji
 - Dostęp do źródła danych
 - Plik hibernate.cfg.xml
 - Pliki mapowania pomiędzy obiektami i tabelkami

Konfiguracja

- Plik hibernate.cfg.xml
 - Można wstawić zawartość bezpośrednio do web.config
 - Można też utworzyć zewnętrzny plik o innej nazwie i dać referencję z web.config

```
<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2" >  
  <session-factory>  
    <property name="connection.driver_class">NHibernate.Driver.SqlClientDriver</property>  
    <property name="connection.connection_string">Server=.;initial catalog=Test;Integrated Security=SSPI</property>  
    <property name="show_sql">>false</property>  
    <property name="dialect">NHibernate.Dialect.MsSql2008Dialect</property>  
  </session-factory>  
</hibernate-configuration>
```

Konfiguracja

- Mapowanie pomiędzy obiektami i tabelkami
 - W zewnętrznych plikach XML
 - Poprzez atrybuty .NET
 - Za pomocą biblioteki Fluent
 - Poprzez Mapping By Code
 - Narzędzia wspierające potrafiące wygenerować
 - Pliki odwzorowań na podstawie bazy danych
 - Schemat bazy danych na podstawie opisu metadanych
 - Pliki klas na podstawie plików odwzorowań
 - Pliki klas na podstawie bazy danych
- Jednak jak i kiedy tego używać?

Konfiguracja mapowania

- Podstawowe elementy
 - Atrybuty
 - Nazwy właściwości, kolumny, typy, operacje (update, insert)
 - Jeśli wielkość litera ma znaczenie → apostrofy
 - Właściwości wyliczane
 - Definiujemy podając atrybut formuła
 - Wartość atrybutu to wyrażenie SQL
 - Dla takiej właściwości nie jest tworzona kolumna w BD

Konfiguracja mapowania

- Podstawowe elementy
 - Tożsamość obiektów
 - Klucz może być jednokolumnowy lub wielokolumnowy
 - Generatory identyfikatorów
 - identity
 - Generuje identyfikatory typów long, short, int
 - Wspiera m.in. DB2, MySQL, MS SQL Server, Sybase
 - sequence
 - Generuje identyfikatory typów long, short, int
 - Wykorzystuje sekwencje m.in. w DB2, PostgreSQL, Oracle, SAP DB

Konfiguracja mapowania

- Generatory identyfikatorów c.d.
 - hilo
 - Generuje identyfikatory typów long, short, int
 - Bazuje na dwóch wartościach hi i lo, dostarczy pakiety Idków
 - guid.comb
 - Generuje GUIDy
 - Sortowanie daje sekwencję dodawania
 - native
 - wybiera jeden z generatorów identity, sequence i hilo w zależności od możliwości bazy danych
- Dobry identyfikator
 - Syntetyczny
 - Uporządkowany
 - Non-insertable

Konfiguracja mapowania

- Podstawowe elementy
 - Komponenty
 - Kilka klas po stronie modelu mapujemy na jedną tabelę po stronie bazy danych
 - Enumeracje
 - Pozwala na przechowywanie czytelnych napisowych wartości po stronie bazy danych

Przykłady

- FactoryCreatingCost
- AttributeMapping
- FluentMappings
- MappingByCodeExample
- UsingComponents
- UsingEnumerations
- CompositeKeys1
- CompositeKeys2
- IdentityMap

Dziedziczenie klas

- Tabela na klasę
- Tabela na hierarchie klas
- Tabela na podklasę

Dziedziczenie klas

- Tabela na klasę
 - Wszystkie właściwości klasy są w tabeli odpowiadającej tej klasie
 - łącznie z właściwościami dziedziczonymi
 - Zaleta: jeśli zapytanie dotyczy jednej klasy, wykona się szybko i łatwo je skonstruować
 - Wada: problem w operacjach polimorficznych

Dziedziczenie klas

- Tabela na hierarchie klas
 - Rozwiązanie polega zastosowaniu jednej tabeli dla hierarchii klas powiązanych relacją dziedziczenia
 - Dodatkowo jest kolumna dyskryminatora, który określa jakiego typu jest dany wpis w tabeli
 - Zaleta: Łatwo zadawać zapytania zwykłe jak i te oparte na polimorfizmie
 - Wady: Zaburzona normalizacja, w każdym wierszu sporo wartości pustych

Dziedziczenie klas

- Tabela na podklasę
 - W tym modelu każdy byt (klasy, w tym abstrakcyjne, interfejsy) mają swoje tabele
 - W tabelach tych są tylko właściwości zdefiniowane w danej klasie lub interfejsie
 - Jeżeli klasa ma podklasę, wtedy jej klucz główny jest jednocześnie obcym do nadklasy i tam znajduje się reszta danych danego obiektu

Dziedziczenie klas

- Tabela na podklasę
 - Zalety
 - Pełna normalizacja schematu w bazie danych
 - Wsparcie dla polimorfizmu
 - Wady
 - Przede wszystkim jedna: przy większej strukturze, duża złożoność obsługi (trzeba wykonywać dużo złączeń)

Dziedziczenie klas

- Wybór strategii
 - W zasadzie wybór jest pomiędzy opcjami 2 i 3
 - Kiedy co wybrać?

Przykłady

- Inheritance1TablePerClass
- Inheritance2TablePerClassHierarchy
- Inheritance3TablePerSubclass

Kolekcje

- Podstawowe rodzaje kolekcji:
 - Set – zbiór
 - Bag – wielozbiór
 - Map – struktura asocjacyjna
 - List – struktura indeksowana, czyli „kolejność ma znaczenie”
- Jakie typy po stronie .NET?
 - Znaczenie biblioteki `lesi.Collection`

Przykłady

- Collections1
- Collections2
- Collections3
- Collections4
- Collections5

Asocjacje

- Reprezentują związki między obiektami, które mają swoje odbicie w związkach między tabelami
- Często są dosyć kłopotliwym elementem do obsłużenia
- Asocjacje mogą być
 - Jednokierunkowe
 - Dwukierunkowe

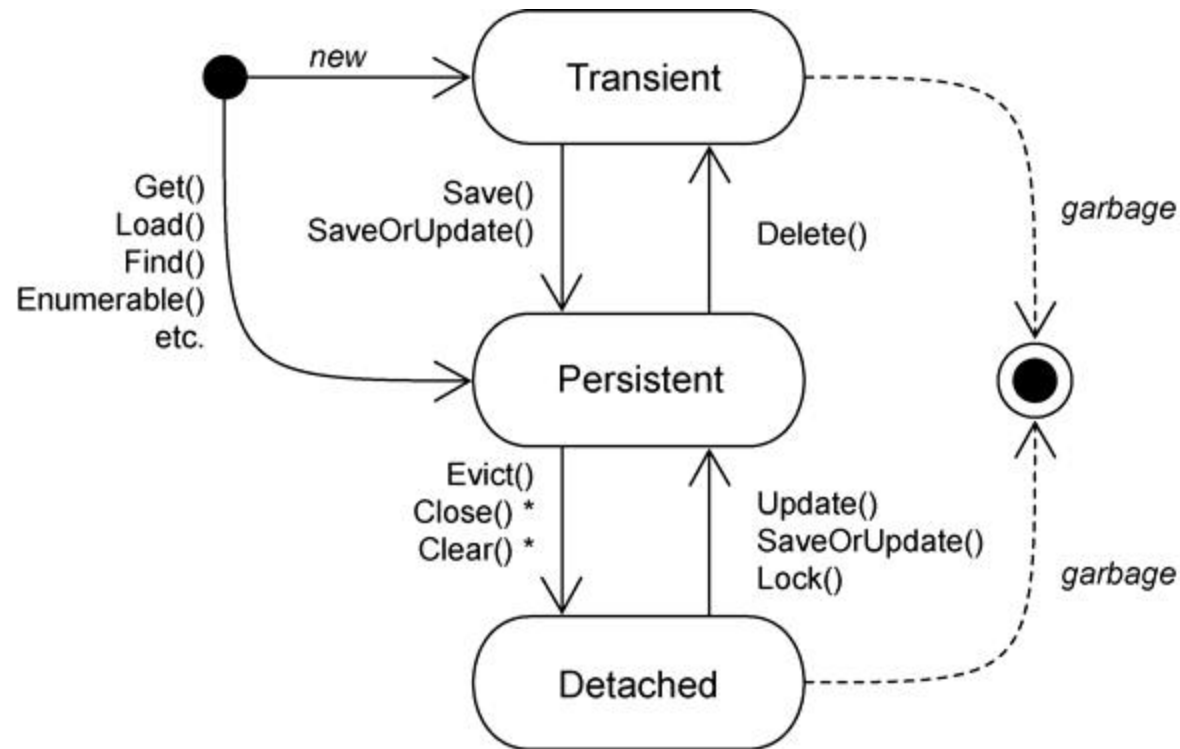
Asocjacje

- Rodzaje asocjacji
 - jeden-do-jednego
 - wiele-do-jednego
 - jeden-do-wielu
 - wiele-do-wielu

Przykłady

- Associations1ManyToOne
- Associations2ManyToMany

Cykl życia obiektów



* affects all instances in a Session

Przykład

- UsingObjects

Trwałość przechodnia

- Realizowana przez opcję cascade
- Bazuje na asocjacjach
- Dostępne wartości
 - none, save-update, delete, all, delete-orphan, all-delete-orphan
- Wartości można łączyć
 - Np. cascade="save-update, delete,,

Przykład

- PersistenceByReachability

Strategie sprowadzania danych

- NHibernate implementuje następujące strategie sprowadzania danych:
 - Sprowadzanie natychmiastowe, ang. *immediate fetching*
 - Sprowadzanie leniwe, ang. *lazy fetching*
 - Sprowadzanie wyprzedzające, ang. *eager (outer join) fetching*
 - Sprowadzanie wsadowe, ang. *batch fetching*
- Do określenia strategii mamy atrybuty w XML:
 - *lazy, extra, fetch, batch-size*
- Ciekawe rozważania:
 - <http://ayende.com/blog/3943/nhibernate-mapping-set>

Strategie sprawozdania danych

- Sprawozdanie leniwe
 - Najczęściej stosowane
 - Ryzyko wiąże się z koniecznością jednoczesnego pobrania, znane jako N+1 antipattern

```
// SELECT * FROM Posts
foreach (Post post in session.CreateQuery("from Post").List())
{
    //lazy loading of comments list causes:
    // SELECT * FROM Comments where PostId = @p0
    foreach (Comment comment in post.Comments)
    {
        //print comment...
    }
}
```

Warto zajrzeć:

<http://www.hibernateingrhinos.com/products/nhprof/learn/alert/selectnplusone>

Strategie sprowadzania danych

- Sprowadzanie wyprzedzające
 - Do pobrania obiektów wykorzystywany jest OUTER JOIN
 - W pewnych scenariuszach jest bardziej optymalne od sprowadzania leniwego
 - Można włączyć na poziomie transakcji, chociaż częściej określa się w pliku mapującym
 - Parametr *max_fetch_depth* w pliku hibernate.cfg.xml określamy, ile tabel może być maksymalnie złączanych
 - domyślnie: 1

Strategie sprowadzania danych

- Sprowadzanie wsadowe
 - Nie jest to osobna strategia sprowadzania, tylko mechanizm przyspieszający działanie sprowadzania leniwego i natychmiastowego.
 - Zamiast podawać w klauzuli WHERE pojedynczy identyfikator, NHibernate zbierze ich więcej i poda raz cały zestaw identyfikatorów

Przykład

- FetchingStrategies

Pobieranie obiektów

- Poprzez identyfikator
- Hibernate Query Language (HQL)
- Query By Criteria (QBC)
- Query By Example (QBE)
- LINQ to NHibernate
- Natywny SQL

Pobieranie obiektów

- Poprzez identyfikator
 - Get
 - `var user = session.Get<User>(userID)`
 - Jeśli nie ma obiektu, zwraca NULL
 - Zwraca zawsze obiekt (nie proxy), nie działa LAZY
 - Load
 - `var user = session.Load<User>(userID)`
 - Jeśli nie ma obiektu, rzuca wyjątek
 - Współpracuje z LAZY

Pobieranie obiektów

- Hibernate Query Language
 - Obiektowy język zapytań podobny do SQL-a
 - Oparty o napisy (ma swoje wady i zalety)
 - Tłumaczony do SQL-a

```
IQuery q = session.createQuery("from User u where u.firstname = :fname");  
q.setString("fname", "Arnold");  
IList<User> result = q.list<User>();
```

Pobieranie obiektów

- Query By Criteria

- Obiektowy język zapytań, kompilowany

```
ICriteria criteria = session.CreateCriteria( typeof( User ) );  
criteria.Add( Expression.Like("firstname", "Alfred") );  
IList<User> result = criteria.List<User>();
```

- Query By Example

- Pozwala na podanie przykładu jako wzorca do wyszukiwania

```
User exampleUser = new User { FirstName = "Alfred" };  
ICriteria criteria = session.CreateCriteria( typeof( User ) );  
criteria.Add( Example.Create(exampleUser) );  
IList<User> result = criteria.List<User>();
```

Pobieranie obiektów

■ Natywny SQL

- `IList<Produkt> produkty =
 session.CreateSQLQuery("select {p.*} from PRODUKT {p}")
 .AddEntity("p", typeof(Produkt)).List<Produkt>();`

■ LINQ to NHibernate

- `from k in session.Query<Koszyk>()
 select k;`

■ LINQ to NHibernate

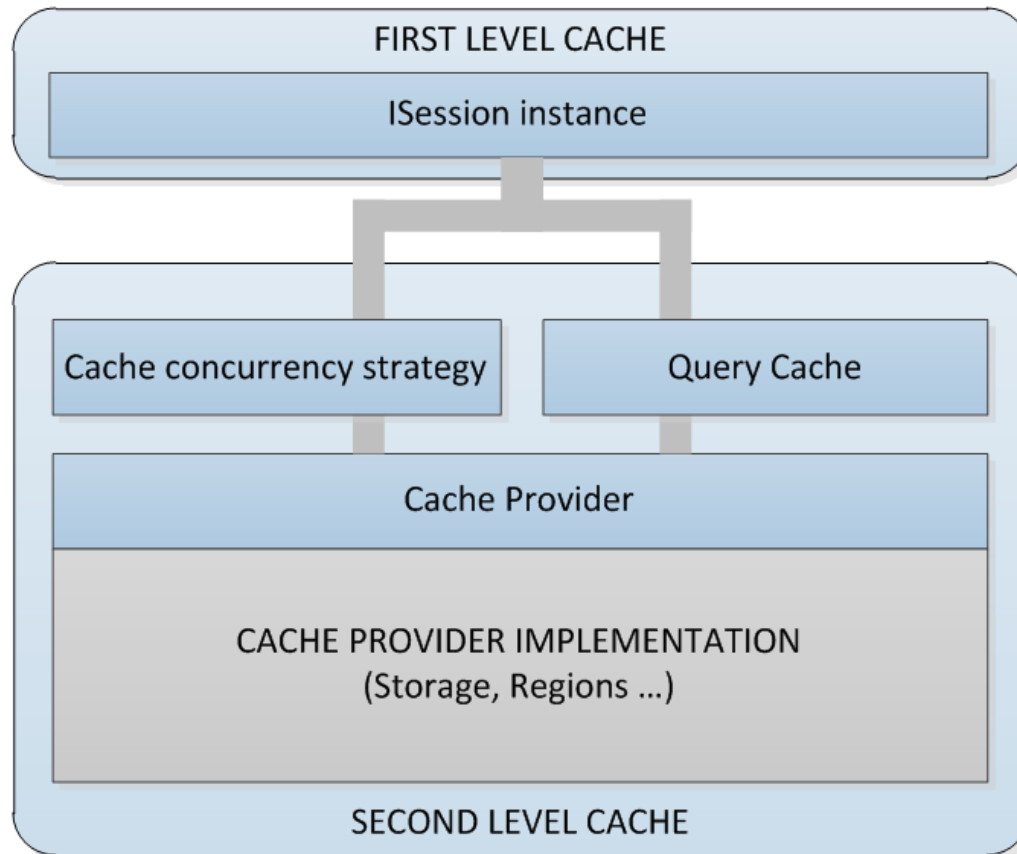
- `session.QueryOver<Koszyk>().List();`

Przykład

- QueryingDatastore

Cache system

- Architektura



Cache system

- Cache poziomu 1
 - Powiązany z obiektem ISession
 - Obiekt znajdzie się w cache po wywołaniu metod:
 - Get, Load, Save, Update, Delete, SaveOrUpdate
Uwaga: zapytania HQL, itd. Nie wstawiają obiektów do cache
 - Cykl życia związany z transakcją oraz następującymi metodami:
 - Session.Clear, Session.Flush, Session.Evict, ...

Cache system

- Cache poziomu 2
 - Powiązany z SessionFactory i dane współdzielone przez wszystkie sesje
 - Dostępnych jest wielu providerów dla cache 2 poziomu:
 - Proces: HashTable, Prevalence, SysCache2
 - Rozproszony: NCache, Velocity
 - Dane pamiętane są w postaci słownika
 - Tutaj mogą być cachowane zapytania, ale tylko ID
 - Mechanizm śledzenia timestampów, generalnie są polityki cache expiration
 - Niektóre providera wspierają SQL Server notifications
 - Cache może być
 - Read-write, Read, Nonstrict-read-write, transactional

Cache system

- Cache poziomu 2, regiony
 - Pozwala na definiowanie obszarów dla zbioru klas
 - Możemy stosować osobne różne ustawienia, np. expiration policy
- Warto poczytać:
 - <http://blog.raffaeu.com/archive/2011/12/29/nhibernate-cache-system-part-1.aspx>
 - <http://blog.raffaeu.com/archive/2011/12/29/nhibernate-cache-system-part-2.aspx>
 - <http://ayende.com/blog/3112/nhibernate-and-the-second-level-cache-tips>

Stateless session

- Stosowane przy masowych operacjach update, delete, save
- Bardziej bezpośredni dostęp do bazy
 - Bez cache 1 i 2 poziomu
 - Nie ma lazy loading
 - Nie ma kaskad
 - Bez śledzenia zmian, transakcje bardzo ograniczone

Debugging

- Drukowanie na konsolę (opcja `show_sql`)
- Logging (NLog)
- SQL Profiler
- NHibernate Profiler