

# Projektowanie aplikacji bazodanowych w .NET

## Wykład 6

Paweł Rajba

Instytut Informatyki  
Uniwersytet Wrocławski

# Plan wykładu

- Wprowadzenie do XPO
- Podstawowe obiekty
- Utrwalanie obiektów
- Transakcje i współbieżność
- Wzorzec *unit of work*
- Odwzorowanie dziedziczenia
- Asocjacje
- Pobieranie danych
- Leniwe pobieranie danych
- Stronicowanie

# Wprowadzenie do XPO

- Produkt firmy Developer Express <http://www.devexpress.com/>
  - jeden z wielu produktów
- Produkt nie jest darmowy (\$199 ze źródłami, \$149 bez źródeł)
- Integracja tylko z .NET
- ORM, w którym aplikacje pisze się bardzo łatwo i szybko
- Bazy danych wspierane przez XPO: Access, Advantage, Asa, Ase, DB2, Firebird, MSAccess, MSSqlServer, MSSqlServerCE, MySQL, Oracle, Pervasive, PostgreSQL, SQLite, VistaDB

# Podstawowe obiekty

## XpoDefault

- Zawiera domyślne ustawienia
- Pozwala korzystać z XPO bez żadnych konfiguracji
  - domyślnie bazą danych będzie lokalnie utworzony plik MDB

## DataLayer

- Odpowiada funkcjonalnością `SessionFactory` z Hibernate
- Jego utworzenie jest kosztowne
- Zawiera konfigurację dostępu do danych oraz „pliki mapujące”
- Mamy dwie podstawowe implementacje
  - `SimpleDataLayer` — pozwala modyfikować metadane podczas działania aplikacji
  - `ThreadSafeDataLayer` — pozwala na dostęp do danych wielu wątkom równocześnie

## Session

- Odpowiada funkcjonalnością `Session` z Hibernate
- Jego utworzenie jest szybkie
  - Przy tworzeniu sesji możemy wskazać odpowiedni `DataLayer`
- Mamy do dyspozycji sesję domyślną `XpoDefault.Session` oraz sesję użytkownika

## XPCollection

- Kolekcja reprezentująca utrwalone obiekty
- Jest powiązana z typem obiektu i z sesją

# Utrwalanie obiektów

- Mamy kilka sposobów na wskazanie, że dane obiekty będą utrwalane:
  - Opatrzanie klasy atrybutem `Persistent`
  - Zdefiniowanie klasy podrzędnej na `XPCustomObject`
  - Zdefiniowanie klasy podrzędnej na `XPObject`
  - Implementacja przez klasę interfejsów `IXPObject`, `IComparable`
- Podstawowe operacje na obiektach
  - `session.Save(o)` — zapisanie lub aktualizacja obiektu
  - `session.Delete(o)` — usunięcie obiektu

## Atrybuty sterujące utrwalaniem

- **PersistentAttribute**
  - Ustawia daną klasę, właściwość lub pole do utrwalenia
  - Podając `Persistent("nazwa")` można określić nazwę, na którą klasa/właściwość/pole będzie mapowane
- **PersistentAliasAttribute**
  - Ustawia właściwość jako ulotną, a jej wartość będzie obliczana z wyrażenia przekazanego do aliasu, opartego na trwałych właściwościach
  - Przykład:

```
[PersistentAlias("UnitPrice*Qty*(1-Discount)")]  
public decimal ExtendedPrice {  
    get { return Convert.ToDecimal(EvaluateAlias("ExtendedPrice")); }  
}
```

## Atrybuty sterujące utrwalaniem c.d.

- `NonPersistentAttribute`
  - Ustawia daną klasę, właściwość lub pole jako ulotne
- `SizeAttribute`
  - Określa rozmiar kolumny tekstowej w bazie danych
  - Podanie jako parametru stałej `SizeAttribute.Unlimited` powoduje utworzenie w bazie danych pola typu CLOB lub TEXT
    - zależy do DBMS



# Przykłady

- DataLayerExample
- PersistenceExample

# Transakcje i współbieżność

- Realizowane są dwa modele współbieżności:
  - Optymistyczna. Model ten uniemożliwia jednoczesną zmianę tego samego obiektu przez dwa procesy.
    - W momencie odczytywania danych obiektu, pobierana jest także wartość systemowego pola `OptimisticLockField` (pola blokady). Przy próbie zapisania zmodyfikowanego rekordu porównywana jest bieżąca wartość pola `OptimisticLockField` z tą pobraną przy wczytywaniu obiektu. Jeśli są różne, rzucany jest wyjątek.
  - Ostatni wygrywa. Wiążąca jest ostatnia aktualizacja rekordu.

# Transakcje i współbieżność

- Transakcje są w ramach sesji. Do obsługi są metody
  - `Session.BeginTransaction()`
  - `Session.CommitTransaction()`
  - `Session.RollbackTransaction()`
- Aby sprawdzić, czy sesja jest w trakcie transakcji:
  - `Session.InTransaction` (read-only property)
- Przykładowa konstrukcja:

```
using (Session session = new Session()) {  
    session.BeginTransaction();  
    try {  
        // Create, update or delete objects  
        session.CommitTransaction();  
    } catch {  
        session.RollbackTransaction();  
        throw;  
    }  
}
```

# Unit of Work

- Zmiany w obiektach są rejestrowane po stronie klienta
- W momencie zatwierdzenia, całość jest wysyłana do bazy d.
- W XPO mamy obiekty `UnitOfWork` i `NestedUnitOfWork`
- Utworzenie `UnitOfWork` powoduje:
  - utworzenie nowej sesji
  - rozpoczęcie nowej transakcji
- Typowe użycie:

```
using ( UnitOfWork uow = new UnitOfWork ) {  
    // Przetwarzanie obiektów  
    uow.CommitChanges();  
}
```

# Unit of Work

- Można też tworzyć zagnieżdżone konstrukcje Unit of Work
- Używamy wtedy obiektu `NestedUnitOfWork`
- Typowa konstrukcja:

```
using ( UnitOfWork uow = new UnitOfWork ) {  
    // Przetwarzanie obiektów  
    using ( NestedUnitOfWork uowNested = uow.BeginNestedUnitOfWork() ) {  
        // Zagnieżdżone przetwarzanie  
        uowNested.CommitChanges();  
    }  
    uow.CommitChanges();  
}
```

- Zmiany są zatwierdzane metodą `uow.CommitChanges()`
  - Wykonanie `CommitChanges()` dla zagnieżdżonego *unit of work* nic tak naprawdę nie robi
  - Zatwierdzenie zmian następuje w zewnętrznym obiekcie `UnitOfWork`.

# Przykłady

- `ConcurrencyExample`
- `TransactionExample`
- `UnitOfWorkExample`

# Odzworowanie dziedziczenia

- XPO realizuje dwa schematy odzworowania:
  - Tabela na hierarchię klas
  - Tabela na każdą klasę (domyślne)
- Do ustawienia sposobu dziedziczenia jest atrybut `MapInheritanceAttribute`, natomiast wartości dostępne są w atrybucie `MapInheritanceTypeAttribute`

# Przykłady

- InheritanceExample1
- InheritanceExample2



# Asocjacje

- Do ustalania relacji służy atrybut `AssociationAttribute`
- Atrybut ma dwa konstruktory
  - `AssociationAttribute(string name)`
  - `AssociationAttribute(string name, Type elemType)`
- Obiekt, który może mieć wiele odwołań do innych obiektów powinien mieć zdefiniowaną kolekcję tych elementów, czyli odpowiedni element `XPCollection`
- Przy zapisywaniu obiektu, który zawiera niezapisane odwołania do innych obiektów, te niezapisane obiekty zostaną automatycznie zapisane
  - Ale niekoniecznie zaktualizowane

## Zapisywanie grafu obiektów

- Domyślne zachowanie przy wywołaniu `o.Save()`
  - zapisywany jest sam obiekt `o`
  - zapisywane są wszystkie nowe obiekty, na które wskazuje obiekt `o`
  - obiekty zmienione, które są wskazywane przez obiekt `o` będą zapisane, jeśli jest uruchomiony trace zmian (w property set jest wywoływana metoda `OnChange`)
- Zachowanie to można zmienić poprzez ustawienie atrybutu `Aggregated` w obiekcie, który ma kolekcję na obiekty zawierające odwołania do tej kolekcji:
  - zapisywany jest sam obiekt `o`
  - zapisywane są wszystkie obiekty, na które wskazuje obiekt `o` (bezwzględnie)

- Realizacja asocjacji jeden-do-wiele

- W klasie *parenta* ustawiamy

```
[Association("ParentChild")]  
public XPCollection<Child> Childs { get { return GetCollection<Child>("Childs"); } }
```

- W klasie *childa* ustawiamy

```
[Association("ParentChild")]  
public Parent ParentId;
```

- Realizacja asocjacji wiele-do-wiele

- W klasie *Location* ustawiamy

```
[Association("LocationsDepartments", typeof(Department))]  
public XPCollection Departments { get { return GetCollection("Departments"); } }
```

- A w klasie *Department*

```
[Association("LocationsDepartments", typeof(Location))]  
public XPCollection Locations { get { return GetCollection("Locations"); } }
```

- W efekcie powstanie tabela łącząca:

Klasa1Kolekcja2\_Klasa2Kolekcja1, czyli

LocationLocations\_DepartmentDepartments

# Przykłady

- AssociationOneToOne
- AssociationOneToManyExample
- AssociationOneToManyExample2
- AssociationOneToManyExample3
- AssociationManyToManyExample

# Pobieranie danych

- W XPO mamy dwa główne sposoby pobierania danych:
  - Za pomocą pseudojęzyka SQL
  - Za pomocą języka obiektowego
  - Poprzez identyfikator obiektu
- Pobierając dane definiujemy kryteria oraz sposoby sortowania
  - Możemy też zdefiniować filtr, czyli kryteria po stronie klienta
- Wyniki zapytań mogą być reprezentowane za pomocą następujących obiektów:
  - XPCollection, XPCursor, XPView
- Kryteria są reprezentowane przez obiekt klasy
  - CriteriaOperator
- Aby utworzyć obiekt kryteriów na podstawie pseudojęzyka odpalamy metodę `CriteriaOperator.Parse()`

## Charakterystyka kolekcji reprezentujących wyniki

- XPCollection
  - Reprezentuje kolekcję trwałych obiektów
  - Istotniejsze właściwości
    - Count
    - Criteria, CriteriaString, Filter
    - SelectDeleted
    - Sorting, TopReturnedObjects
  - Tworzenie obiektu:
    - `XPCollection<Beer> xp = new XPCollection<Beer>(session);`
    - `XPCollection xp = new XPCollection(session, typeof(Example.Beer));`

## Charakterystyka kolekcji reprezentujących wyniki

- XPCursor
  - Nie pobiera wszystkiego za jednym razem, tylko po kawałku
    - ten kawałek jest określany przez parametr `PageSize`
  - Istotniejsze właściwości
    - `Count`, `PageSize`, `SelectDeleted`,
    - `Sorting`, `TopReturnedObjects`
  - Kryteria zapytania możemy określić w wybranych konstruktorach podczas tworzenia obiektu
  - XPCursor jest ukierunkowany na optymalne wykorzystanie pamięci, przez co jest wolniejszy.
    - Teoretycznie XPCollection jest znacznie szybszy, ale w testach wyszło, że XPCursor jest tylko nieznacznie wolniejszy.

## Charakterystyka kolekcji reprezentujących wyniki

- XPView
  - Reprezentuje wynik zapytania SQL
  - Jest w trybie „tylko-do-odczytu”
  - Dostępne właściwości są definiowane samodzielnie
    - reprezentowane przez właściwość `Properties`
- Istotniejsze właściwości i metody
  - `Count`, `Criteria`, `CriteriaString`, `Filter`
  - `Properties`, `Sorting`, `TopReturnedRecords`
  - `AddProperty()`
- Klasy `ViewProperty` i `ViewRecord`



## Klasy do budowania obiektowych zapytań

- `AggregateOperand` — reprezentuje funkcje agregujące
  - MIN, MAX, SUM, itd.
- `BetweenOperator` — reprezentuje operator BETWEEN
- `BinaryOperator` — służy do porównywania dwóch wartości
- `GroupOperator` — reprezentuje operatory AND i OR
- `InOperator` — reprezentuje operator IN
- `UnaryOperator` — reprezentuje operatory jednoargumentowe
  - NOT, ISNULL, itd.
- `OperandProperty` — reprezentuje właściwość
- `OperandValue` — reprezentuje wartość

## Sortowanie

- Do definiowania sortowania mamy klasę `SortProperty`
- Istotniejsze właściwości klasy `SortProperty`
  - `PropertyName`
  - `Direction` o wartościach
    - `DevExpress.Xpo.DB.SortingDirection.Ascending`
    - `DevExpress.Xpo.DB.SortingDirection.Descending`
- Do metod pobierających dane zwykle przekazuje się:
  - Pojedynczy obiekt lub tablicę obiektów `SortProperty`
  - Obiekt `SortingCollection`
    - Tworzymy go i wypełniamy za pomocą schematu:

```
SortingCollection sc = new SortingCollection();  
sc.Add( sortProperty1 ); sc.Add( sortProperty2 ); ...
```

# Przykłady

- QueryingDatastore1
- QueryingDatastore2

# Leniwe pobieranie danych

- Przy wczytywaniu obiektu, domyślnie wczytywane są wszystkie jego pola
  - Dotyczy to także pól kluczy obcych (ale nie dotyczy kolekcji)
- To domyślne zachowanie można zmienić stosując technikę leniwego pobierania danych
- Przykładowy schemat właściwości leniwie pobieranej:

```
private XPDelayedProperty description = new XPDelayedProperty();
[Delayed("description")]
public string Description
{
    get { return Convert.ToString( description.Value ); }
    set { description.Value = value; }
}
```

- Technikę możemy także stosować do pól kluczy obcych

# Przykład

- LazyLoading

# Stronicowanie

- Wczytywanie ze stronicowaniem należy wykonywać za pomocą `XPCursor` lub `XPCollection`
- Użycie `XPCursor` jest kosztowne, gdy chcemy pobrać jedną z ostatnich stron
- Alternatywą jest stronicowanie istniejącego `XPCollection`
  - Odbywa się z wykorzystaniem `XPageSelector`
  - Ustawiamy
    - Powiązanie z odpowiednią kolekcją `XPCollection` oraz
    - `PageSize` (rozmiar strony) i `CurrentPage` (numer strony)
  - W `XPCollection` widoczne są tylko rekordy określone przez ustawione parametry.
    - Z bazydanych pobierane są identyfikatory wszystkich rekordów
    - Następnie pobierane są dane wybranej strony

# Przykład

- PagingExample

# LINQ to XPO

- Obecnie XPO daje wsparcie dla LINQ
- Niestety nadal nie są zaimplementowane wszystkie operatory, m.in. dosyć ważny operator *Skip*
- W punktu widzenia składni użycie LINQ w XPO jest bardzo łatwe, przykładowa konstrukcja:

```
XpoDefault.ConnectionString = "ConnectionString";  
var customers = new XPQuery<Customer>( Session.DefaultSession );  
var list =  
    from c in customers  
    where ( c.Country == "Germany"  
           && c.ContactTitle == "Sales Representative" )  
    orderby c.ContactName  
    select c;  
foreach ( Customer cust in list )  
    Console.WriteLine( string.Format( "{0}\t{1}\t{2}",  
                                     cust.ContactName, cust.Country, cust.ContactTitle ) );
```



# Przykład

- LinqToXpoExample