Projektowanie aplikacji z bazami danych

Systemy mapowania relacyjno-obiektowego

Paweł Rajba

Instytut Informatyki

Uniwersytet Wrocławski

Plan wykładu

- Wprowadzenie do trwałości
- Niedopasowanie paradygmatów
- Architektura warstwowa
- Czym jest ORM?
- Problemy i pytania związane z ORM
- Dlaczego warto użyć ORM?
- Kupić czy napisać?
- Systemy ORM

Czym jest trwałość?

- Składowanie danych utworzonych podczas działania programu
  - Najczęściej obecnie w bazie relacyjnej
- Relacyjne bazy danych są najbardziej rozpowszechnione
  - Były próby stworzenia obiektowych baz danych, ale na razie niewiele z tego wyszło
- „Prosty schemat" uzyskania trwałości
  - generujemy przygotowane zapytania dla obiektów
  - parametryzujemy zapytania dla konkretnych obiektów, przeglądamy wyniki zapytań,...

Czym jest trwałość?

- „Prosty” schemat realizuje dostęp niskopoziomowy
  - Zadanie dosyć żmudne
  - Programista powinien się skupić na tworzeniu logiki biznesowej, a nie obsługą trwałości
  - To może zrezygnować z baz relacyjnych?
    — Niestety, jedyne sprawdzone rozwiązanie
- 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.

Czym jest trwałość?

- Pojęcie obiektowego modelu dziedziny
  - abstrakcja identyfikująca rzeczywiste byty występujące w danej dziedzinie i powiązania między nimi
  - nie jest przeznaczony dla programistów i nie zawiera szczegółów dotyczących implementacji
- Systemy ORM operują na modelu dziedzinowym
- Uwaga: proste i małe aplikacje czasami łatwiej napisać bez korzystania z modelu dziedzinowego i ORM, tylko po prostu operując na tabelkach

Niedopasowanie paradygmatów

W czym rzecz

- Polega na zupełnie innych filozofii dotyczących modelu relacyjnego i obiektowego
- Niedopasowanie implikuje szereg konkretnych problemów

Problem szczegółowości

- Przyjrzyjmy się mu na przykładzie. 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.

Niedopasowanie paradygmatów

Problem podtypów

- W większości (czyli we wszystkich) DBMS nie jest obsługiwane dziedziczenie
- Z drugiej strony dziedziczenie to podstawowy mechanizm w językach obiektowych
- Zagadnienie asocjacji polimorficznej. Rozpatrzmy przykład:
  - Mamy klasy User $\overset{1..*}{\text{---}}$ Payment,

    CreditCard $\rightarrow$ Payment, BankAccount $\rightarrow$ Payment
  - Powiązanie User–Payment realizuje asocjację polimorficzną

Niedopasowanie paradygmatów

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

Niedopasowanie paradygmatów

Problemy dotyczące asocjacji

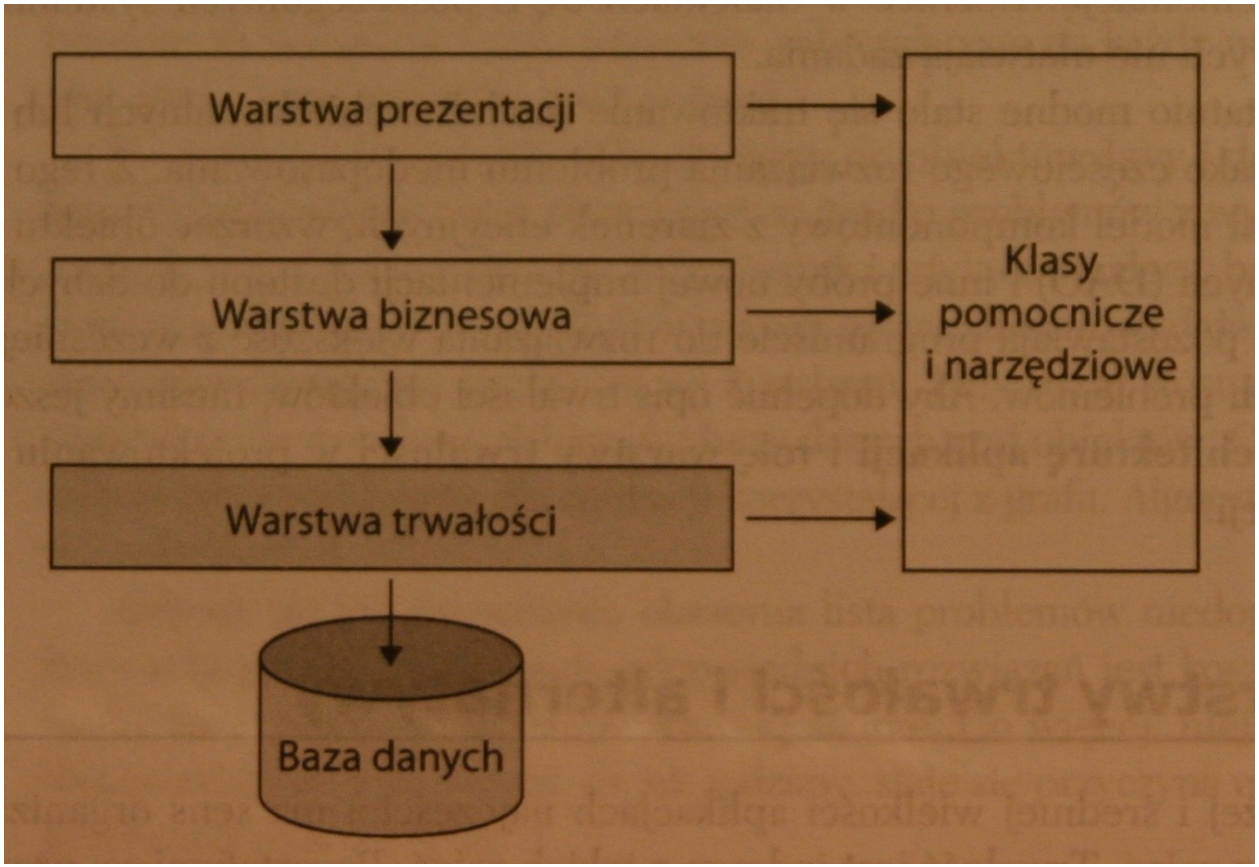
- W systemie relacyjnym mamy związki jeden-do-wielu i jeden-do-jednego
  - Związek wiele-do-wielu jest tak naprawdę połączeniem związków jeden-do-wielu
- W świecie obiektowym poprzez kompozycję możemy z kolei
  - tworzyć asocjacje jednokierunkową
  - tworzyć asocjacje dwukierunkową, definiując odpowiednie 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

Niedopasowanie paradygmatów

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 trochę, czyli bardzo nieefektywnie
- Ogólnie pojawia sie problem odwzorowań języka wewnętrznego systemu ORM na odpowiedni dialekt SQL

Architektura warstwowa



Źródło: C. Bauer, G. King, Hibernate in Action, Manning 2005

Architektura warstwowa

Co w warstwie trwałości?

- Serializacja
  - Jest to podstawowy, wbudowany w język mechanizm trwałości
  - Klasa jest zdolna do trwałości, gdy dodamy atrybut Serializable
    - W przypadku serializacji XML nawet to nie jest potrzebne
  - Zapisywane są automatycznie całe grafy obiektów
  - Przydaje się głównie w
    - prostych zastosowaniach np.rejestracji profilu użytkownika,
    - wymianie komunikatów w protokołach wykorzystujących Internet (ale nie tylko), np. w komunikacji z WebService

Architektura warstwowa

Co w warstwie trwałości?

- Dlaczego nie serializacja? Mechanizm ten nie umożliwia
  - Formułowania zapytań
    - Nie możemy wczytać obiektu warunkowo
  - Częściowego odczytu bądź aktualizacji
    - Co jest przydatne przy operowaniu na dużych danych
  - Zarządzania cyklem życia obiektów
    - Nie ma tutaj pojęcia stanu
  - Współbieżności i transakcji
    - Nie można równocześnie czytać i zapisywać z tego samego strumienia (np. poprzez wątki)
    - Pojęcia transakcji w ogóle w przypadku serializacji nie ma

Architektura warstwowa

Co w warstwie trwałości?

- Ręczne tworzenie kodu za pomocą ADO.NET
  - Zaleta: mamy dokładnie to co chcemy
  - Wada: kupa roboty i związanych z tym kosztów
    - a po co, skoro są gotowe rozwiązania
- Odwzorowanie obiektowo–relacyjne
  - W pewnym rodzaju problemów/zadań najlepsze rozwiązanie
  - Nie są to rozwiązania pozbawione wad

Odwzorowanie obiektowo–relacyjne

Czym jest ORM?

- Oprogramowanie zapewniający trwałość automatycznie
- Automat translacji działa na podstawie metadanych opisujących odwzorowanie obiektu na dane
- Translacja jest przezroczysta i działa w obie strony

Cztery główne składowe ORM

- interfejs pozwalający na wykonywanie operacji CRUD na obiektach klas umiejących zapewnić trwałość
- interfejs lub język pozwalający zadawać zapytania
- narzędzia do określania metadanych
- elementy dodatkowe: obsługa transakcji, leniwe pobieranie asocjacji, optymalizacje

Problemy i pytania związane z 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 sa zagadnienia:
  - atrybut „not null”,
  - dostępność pól: public, private, protected,
  - nazewnictwo
    - np. w Oracle nazwy mogą mieć co najwyżej 30 znaków

Problemy i pytania związane z ORM

- W jaki sposób realizowana jest tożsamość obiektów?
- Jak tworzyć obiekt logiki biznesowej (user, payment, itd.)?

  Czy można to automatyzować?
- Jak wygląda współpraca pomiędzy obiekatmi logiki biznesowej a obiektami oprogramowania ORM?
- Jakie są możliwości języka zapytań?
- Jak wydajne jest pobieranie danych z asocjacji?
  - nasz przykład: jednostka.getOsoba(3).getUlica()

Dlaczego warto użyć ORM?

Rozwiązanie oparte na ORM zapewnia:

- Produktywność
  - programista skupia się na problemie biznesowym, a nie składowaniem obiektów
- Konserwację
  - Mniej kodu przez co łatwiej panować nad aplikacją
  - ORM jest zwykle bardziej elastyczny niż własna warstwa dostępu do danych, przez co łatwiej modyfikować aplikację

Dlaczego warto użyć ORM?

Rozwiązanie oparte na ORM zapewnia c.d.:

- 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 dopracowany
  - Często też istotną kwestią określającą wydajność ORM jest odpowiednia konfiguracja
- Niezależność od dostawcy
  - Jedna z istotniejszych zalet: zwykle raz napisana aplikacja będzie działać na Oracle, SQL Server, PostgreSQL, itd.

Istotne pytanie: kupić czy napisać?

- Napisanie rozbudowanego systemu ORM nie jest trywialne i jest czasochłonne
- Odpowiedź na pytanie zależy przede wszystkim od
  - Czasu, który jest na „zdobycie" systemu
  - Wymagań, które system powinien spełniać
  - Kosztów, które można ponieść
- Odpowiedź ułatwiają ORM-y udostępniane na licencji GPL
- Zwykle odpowiedź brzmi: kupić lub wykorzystać jeden z darmowych

Jakie ORM będziemy omawiać?

- NHibernate
- ADO.NET Entity Framework