

# Mapowanie relacyjno-obiektowe

## Projektowanie i programowanie systemów internetowych I

---

mgr inż. Krzysztof Rewak

22 kwietnia 2018

Wydział Nauk Technicznych i Ekonomicznych

Państwowa Wyższa Szkoła Zawodowa im. Witelona w Legnicy

# Plan prezentacji

1. Bazy danych w aplikacjach webowych
2. Mapowanie z relacji na obiekty
3. Wzorce
4. Przykłady zastosowań
5. Podsumowanie

# Bazy danych w aplikacjach webowych

---

# Operacje na bazach danych

```
$connection_string = "mysql:host=localhost;dbname=db";  
$user = "root";  
$password = "";  
  
$pdo = new \PDO($connection_string, $user, $password);  
$users = $pdo->query("SELECT * FROM users;");  
  
foreach($users as $user) {  
    echo $user;  
}
```

# Operacje na bazach danych

```
import MySQLdb as m

host = "localhost"
user = "root"
password = ""
db = "db"

db = m.connect(host=host, user=user, passwd=password, db=db)

cursor = db.cursor()
cursor.execute("SELECT * FROM users;")

users = cursor.fetchall()

for user in users:
    print user

db.close()
```

# Operacje na bazach danych

```
var connection = InitializeConnectionToDb();  
var select = new SelectBD();  
  
select.Open(connection, @"SELECT * FROM Users");  
foreach(var user in select.DataTable)  
{  
    Console.WriteLine(user);  
}
```

Czy można zbudować wielki system internetowy korzystając z prostego wypisywania kwerend, wysyłania ich do bazy danych i interpretacji wyników?

Czy można zbudować wielki system internetowy korzystając z prostego wypisywania kwerend, wysyłania ich do bazy danych i interpretacji wyników?

Oczywiście, że można.



Czy można sensownie zbudować wielki system internetowy korzystając z prostego wypisywania kwerend, wysyłania ich do bazy danych i interpretacji wyników?

Czy można sensownie zbudować wielki system internetowy korzystając z prostego wypisywania kwerend, wysyłania ich do bazy danych i interpretacji wyników?

Może niekoniecznie.

`SELECT content FROM disadvantages;`

Wady korzystania z ręcznego budowania zapytań:

- redundancja kodu;
- brak kontroli na spójnością systemu;
- problemy przy zmianach koncepcji;
- wymóg poznania przynajmniej dwóch języków programowania?

## Mapowanie z relacji na obiekty

---

# Uproszczony schemat działania ORM



# Spaghetti code

```
$query = "SELECT id, login, email  
        FROM users  
        WHERE id = 1;  
        ";  
  
$result = $pdo->query($query);
```

# Spaghetti code

```
$query = "SELECT users.id, login, email, birthdate
          FROM users
          JOIN profiles
            ON profiles.id = users.id
          WHERE users.id = 1;
          ";

$result = $pdo->query($query);
```

# Lepiej?

```
User::with("profiles")->find(1);
```



## Co warto zapamiętać?

ORM jest systemem mapującym - a więc tłumaczącym - bazodanowe tabele i relacje na model obiektowy.

**Nie jest to w żadnym wypadku obiektowa baza danych**, a jedynie sposób, aby wykorzystać nieobiekтовую bazę w obiektowy sposób.

Dlaczego warto korzystać z mapowania relacyjno-obiektowego?

# Spójność danych

Zaletą: Odpowiednio zamodelowane dane będą spójne względem siebie i systemu.

# Spójność danych

Przykład: klient ma sklep internetowy. Jak wprowadzić do działającego już systemu kilka walut, bo klient chce wejść na rynek międzynarodowy?

Jedno z rozwiązań: przepisać wszystkie zapytania na joiny do tabeli walut.

Jedno z rozwiązań: zmienić metodę pobierania produktów w modelu na pobieranie z połączeniem do walut i ewentualnym ustawieniem waluty domyślnej.

# Separacja warstw

Zaleta: Można oddzielić warstwę logiki biznesowej od warstwy zapisanych danych.

# Separacja warstw

Przykład: klient ma system do zarządzania przedsiębiorstwem. Jak sensownie oddzielić dane pracowników obecnie pracujących od tych, którzy już nie pracują?

## Separacja warstw

Jedno z rozwiązań: stworzyć dwie tabele `active_employees` i `nonactive_employees` lub dodać kolumnę `is_active` do łączonej tabeli `employees`.

Jedno z rozwiązań: stworzyć modele `ActiveEmployee`, `NonactiveEmployee` i `Employee`, które będą bazowały na jednej tabeli.



## Zależności technologiczne

Zalety: Można uniezależnić się od technologii bazodanowej.

## Zależności technologiczne

Przykład: klient ma system do zarządzania budżetem. Okazuje się, że jego silnik bazy danych ma w sobie krytyczny błąd bezpieczeństwa, więc trzeba relatywnie bezkosztowo zmienić DBMS. Co robić?

# Zależności technologiczne

Jedno z rozwiązań: przepisać dosłownie wszystko.

Jedno z rozwiązań: zmienić wartość jednej zmiennej konfiguracyjnej.

Więcej zalet?

- większa kontrola nad kodem,
- praca nad konkretnymi obiektami, a nie abstrakcyjnymi tablicami,
- możliwość podpięcia obserwatorów/zdarzeń,
- możliwość podpięcia systemu cache'ującego,
- wiele innych.

**Wzorce**

---

# Jak ugryźć ORM?

Warto omówić przynajmniej dwa podstawowe podejścia do mapowania relacyjno-obiektowego. Oba wykorzystują wzorce architektoniczne: kolejno *active record* i *data mapper*.

# Aktywny rekord

*Aktywny rekord* to - uogólniając - pomysł wedle którego każdy obiekt wykorzystywany przez ORM jest wyposażony w najważniejsze funkcjonalności.

# Aktywny rekord

Każdy obiekt powinien dziedziczyć po nadrzędnej klasie *modelu*, która umożliwi pracę w formie aktywnego rekordu.

Przykładowy model najprostszego produktu sklepowego może wyglądać następująco:

```
class Product(models.Model):  
    name = models.CharField(max_length=64)  
    price = models.DecimalField()
```



# Aktywny rekord

Klasę modelu można wówczas wykorzystać między innymi do tworzenia obiektów i zapisywania ich w bazie danych:

```
product = Product(name="Infinity Gauntlet", price=99.90)
product.save()
```

# Aktywny rekord

Klasa modelu może też służyć jako źródło do pobierania obiektów z bazy:

```
products = Product.objects.all()
```

```
for product in products:  
    print product
```

# Aktywny rekord

Klasa modelu może też służyć jako źródło do pobierania obiektów z bazy:

```
products = Product.objects.filter(price__lte=100)

for product in products:
    print product
```

# Aktywny rekord

Przykładowo można usunąć wszystkich użytkowników, którzy nie aktywowali się przeciągu tygodnia od rejestracji:

```
from datetime import datetime, timedelta

threshold = datetime.today() - timedelta(days=7)

users = User.objects.filter(created_at__lte=threshold,
                             is_active=False)

for user in users:
    user.delete()

# OR users.delete()
```

# Aktywny rekord

Zalety?

- wszystko jest *pod ręką*,
- początkowo może się wydawać bardziej intuicyjny.

Wady?

- ciężkie modele,
- brak separacji warstw.

# Konwerter danych

*Konwerter danych* to - uogólniając - pomysł wedle którego każdy obiekt wykorzystywany przez ORM jest jedynie reprezentacją danych, a funkcjonalnościami zajmuje się osobna warstwa.

# Konwerter danych

Obiekt to tylko i wyłącznie rozpisane dane:

```
class Product():  
    name = None  
    price = None
```

# Konwerter danych

Obiekt modelu można wówczas przekazać do specjalnego menadżera, który umożliwi zapisywanie obiektów w bazie danych:

```
manager = Manager()

product = Product()
product.name = "Infinity Gauntlet"
product.price = 99.90

manager.persist(product)
manager.flush()
```



# Konwerter danych

Z kolei specjalne repozytorium może służyć jako źródło do pobierania obiektów z bazy:

```
repository = Repository(Product)

products = repository.all()

for product in products:
    print product
```

# Konwerter danych

Z kolei specjalne repozytorium może służyć jako źródło do pobierania obiektów z bazy:

```
repository = Repository(Product)

products = repository.filter("price", "lte", 100)

for product in products:
    print product
```

# Konwerter danych

Przykładowo można usunąć wszystkich użytkowników, którzy nie aktywowali się przeciągu tygodnia od rejestracji:

```
from datetime import datetime, timedelta

threshold = datetime.today() - timedelta(days=7)

manager = Manager()
repository = Repository(Product)

products = repository.filter("created_at", "lte", threshold)
                        .filter("is_active", "=", False)

for product in products:
    manager.delete(product)
    manager.flush()
```

Zalety?

- zgodny między innymi z zasadą pojedynczej odpowiedzialności,
- mniej kodu: lżejsze modele i inicjowane repozytoria.

Wady?

- trudniejszy do opanowania i implementacji

## Przykłady zastosowań

---

Popularne systemy ORM w PHP:

- Doctrine ORM
- Eloquent ORM
- RedBean ORM
- Propel

Przykład wybrania danych przez Doctrine:

```
public function all()
{
    $people = $this->getDoctrine()
        ->getRepository(PersonEntity::class)
        ->findAll();

    return $this->render("person/list.twig", [
        "people" => $people
    ]);
}
```

Popularne systemy ORM w Pythonie:

- SQLAlchemy
- Django ORM



Popularne systemy ORM w C#:

- Entity Framework
- NHibernate
- Dapper

Popularne systemy ORM w C#:

- Hibernate
- jOOQ
- ActiveJDBC

# Podsumowanie

---

# Podsumowanie

- ORM to system mapowania bazy danych na obiekty,
- ORM *tłumaczy* łańcuchy metod wywoływane na obiekcie na wybrany dialekt SQL,
- ORM są wygodne w użyciu dla programistów,
- ORM są popularne i eliminują wiele błędów, które mogą popełnić początkujący programiści.

Ale niestety:

- ORM to dodatkowa warstwa abstrakcji, więc łatwo można *przedobrzyć* i skomplikować z pozoru łatwe rzeczy,
- korzystanie z ORM może doprowadzić do błędów wydajnościowych (choćby problem  $n + 1$  zapytań),
- każdy ORM działa inaczej, więc mimo wszystko wymagane jest poznanie nowego narzędzia.

# Bibliografia i ciekawe źródła



<http://designpatternsphp.readthedocs.io/pl/latest/Structural/DataMapper/README.html>



<https://docs.djangoproject.com/en/2.0/topics/db/models/>



<https://www.doctrine-project.org/projects/orm.html>

**Pytania?**

Kod prezentacji dostępny jest w repozytorium git pod adresem  
<https://bitbucket.org/krewak/pwsz-ppsi>



Wszystkie informacje dot. kursu dostępne są pod adresem  
<http://pwsz.rewak.pl/kursy/4>

