Politechnika Wrocławska

Internetowe Bazy Danych

dr inż. Roman Ptak
Katedra Informatyki Technicznej
roman.ptak@pwr.edu.pl

Plan wykładu 4.

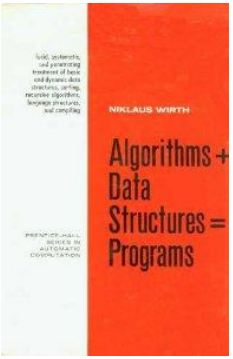
- Wzorzec MVC
- Języki skryptowe: Python, Ruby
- Frameworki MVC i inne
- Obiektowy model danych a bazy danych
- ORM – mapowanie obiektowo-relacyjne

Wzorce architektury, projektowe, …

MVC

Pojęcia inżynierii oprogramowania

- Algorytmy + struktury danych = programy
  Niklaus Wirth, *Algorithms + Data Structures = Programs*, 1976

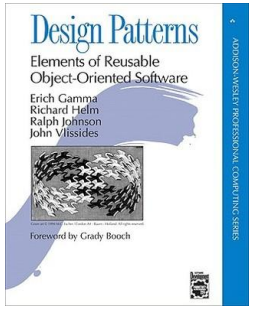


Pojęcia inżynierii oprogramowania (2)

- Wzorce projektowe

  Erich Gamma, Ralph Johnson, John Vlissides, Richard Helm, *Design Patterns: Elements of Reusable Object-Oriented Software*, 1994

„Zaproponowane przez Kenta Becka oraz Warda Cunninghama w 1987 roku, dogłębnie opracowane i rozpropagowane przez tzw. Bandę Czterech (Gang of Four) szybko stały się podstawowym narzędziem w rękach analityka, projektanta, programisty systemów IT”*.



* http://analizait.pl/2013/wstep-do-wzorcow-projektowych/

Kategoryzacja wzorców

- Wzorce analizy – **poziom opisu rzeczywistości**
- Wzorce architektury (ang. *architectural patterns*) – dotyczą całego systemu informatycznego bądź też jego modułów – **poziom integracji komponentów**, np. wzorzec MVC
- Wzorce projektowe (ang. *design patterns*) – dotyczą problemów z komunikacją, tworzeniem lub organizacją obiektów i klas – **poziom interakcji pomiędzy klasami**
- Wzorce implementacyjne – **poziom języka programowania**

Wzorce projektowe

- Wzorce projektowe – uniwersalne, sprawdzone w praktyce rozwiązania często pojawiających się, powtarzalnych problemów projektowych.
- Pokazują powiązania i zależności pomiędzy klasami oraz obiektami.
- Ułatwiają tworzenie, modyfikację oraz pielęgnację kodu źródłowego.
- Są opisami rozwiązania, a nie jego implementacją.
- Stosowane w projektach wykorzystujących programowanie obiektowe.

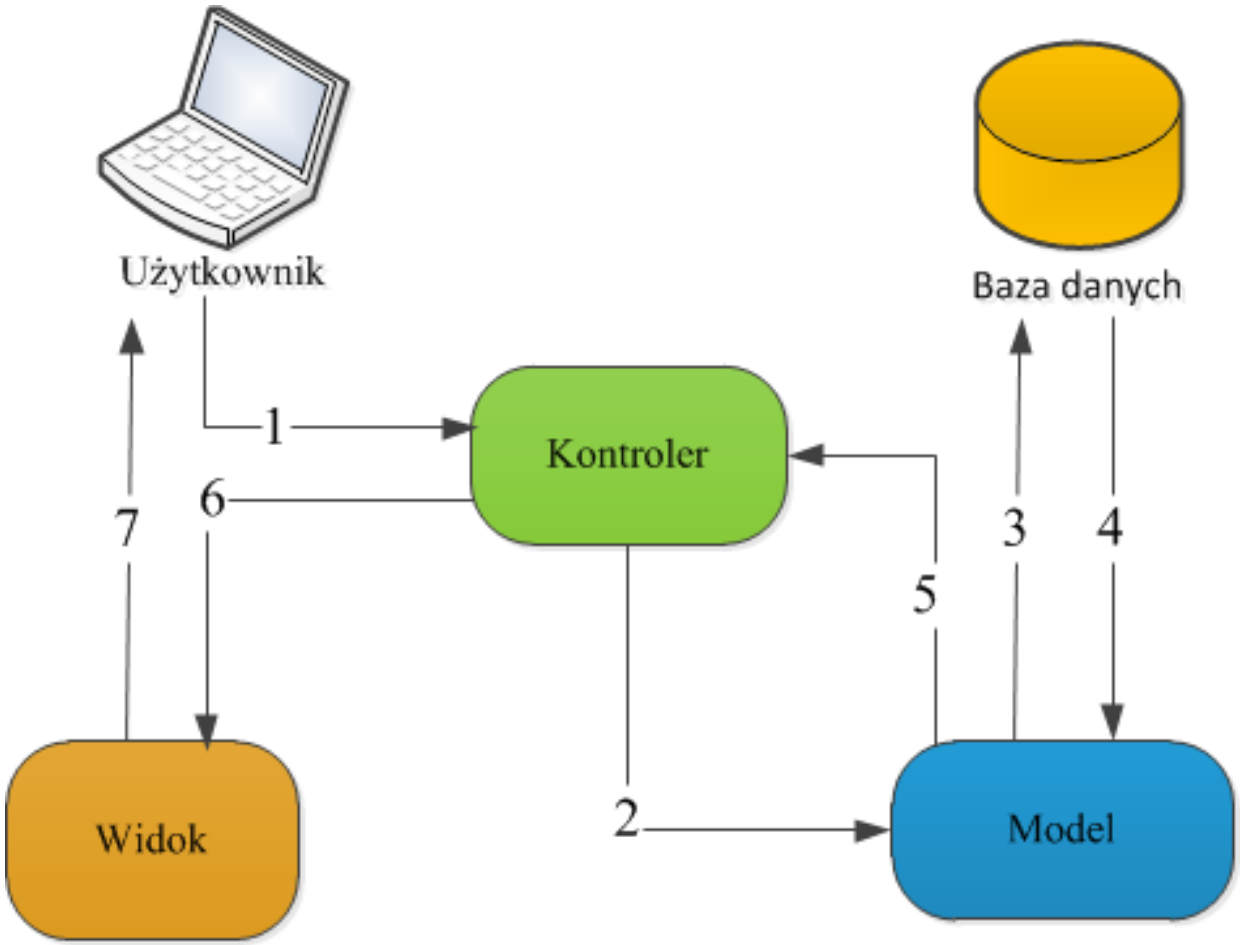
Klasyfikacja wzorców projektowych

- Wzorce konstrukcyjne
- Wzorce strukturalne
- Wzorce czynnościowe

MVC

- Model-View-Controller – wzorzec architektoniczny służący do organizowania struktury aplikacji posiadających graficzne interfejsy użytkownika.
- Może być rozumiany jako złożony wzorzec wykorzystujący idee wzorców prostych:
  - Obserwator (w modelu i widoku),
  - Strategia (w widoku i kontrolerze),
  - Kompozyt (w widoku).

Schemat działania MVC



MVC w aplikacjach internetowych

- W tradycyjnych aplikacjach z interfejsem MVC jest sposobem na dobrą organizację kodu (jeden język programowania).
- Odzwierciedla architekturę aplikacji internetowych, które dzielą się na część kliencką i serwerową.
- Wielojęzyczność komponentów aplikacji internetowych.

Przykładowe elementy architektury MVC

Na podstawie: Edward Benson, *Rails. Sztuka programowania*, 2009

	Model	Widok	Kontroler
Co?	Obiekty reprezentujące dziedzinę aplikacji internetowej. Stanowią otoczkę tabel bazy danych.	Zbiór szablonów HTML, arkuszy CSS i kodu JavaScriptu, który zapewnia odpowiedni wygląd i działanie witryny po stronie użytkownika.	Kod przyjmujący, żądania, modyfikujący model i renderujący widok wysłany do użytkownika.
Przykłady	Użytkownik Artykuł Komentarz Aukcja Zdjęcie	Strona profilu użytkownika. Artykuł na blogu. Formularz dodawania komentarza. Strona szczegółów aukcji. Miniaturka zdjęcia.	Kontroler konta zarządzający rejestracją, logowaniem i obsługą zgubionych haseł.
Języki	PHP, Ruby, Java itp. dla reprezentacji obiektów; SQL dla trwałości danych	HTML, CSS, JavaScript	PHP, Ruby, Java itp.

Przykładowe implementacje MVC

- **Platforma JEE**
  - Kontroler – klasy *serwletu*.
  - Model – klasyczne klasy Javy wspierające dostęp do baz danych za pomocą *ODBC*.
  - Widok – *dokumenty JSP*, wspomagane HTML, CSS i innymi podobnymi technologiami.
- **PHP i Smarty**
  - Kontroler – skrypty PHP.
  - Model – obiektowe klasy języka PHP.
  - Widok – *szablony Smarty*, wspomagane HTML, CSS i innymi podobnymi technologiami.

Własności MVC

- Model jest reprezentacją pewnego fragmentu rzeczywistości. W danej aplikacji może istnieć kilka modeli tego samego elementu.
- We współczesnych aplikacjach modele mają postać klas udostępniających pewien interfejs programistyczny do manipulacji strukturami danych oraz wykonywania pewnych akcji.
- Widoki przechowują cały kod HTML strony wraz ze stylami i wykorzystywane są do wyświetlania przekazanych danych
- Zadaniem kontrolera jest odbiór, przetworzenie oraz analiza danych wejściowych od użytkownika.

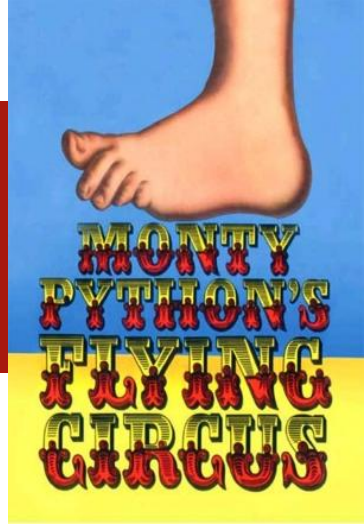
Wzorce pochodne

- Model-View-Presenter (MVP)
- Presentation-Abstraction-Control
- Hierarchical Model-View-Controller
- Pasywny widok
- Model View ViewModel (MVVM)

PHP, Python, Ruby

JĘZYK SKRYPTOWE

Język Python



- Dynamiczny, obiektowy język programowania wysokiego poziomu ogólnego przeznaczenia
- Istnieje od 1991 r.
- Aktualna wersja stabilna: 3.5.2
  - wcześniejsza: 2.7.12
- Oficjalna strona: https://www.python.org

python™

Cechy Pythona

- Czysta, łatwa do czytania składnia
- Intuicyjne programowanie obiektowe, naturalny sposób wyrażania kodu proceduralnego
- Automatyczne zarządzanie pamięcią
- Pełna modułowość, wsparcie dla hierarchicznych pakietów
- Typy danych wysokiego poziomu
- Oparty na wyjątkach system obsługi bledów
- Wiele dostępnych bibliotek
- Przenośny na różne platformy sprzętowe

Python: Hello world

- plik hello_world.py

```
#!/usr/bin/python

# Hello world python program

print "Hello World!";
```

Wcięcia jako element strukturalny

- Przykłady

```
x = 1
if x == 1:
    print "x is 1."

def silnia(x):
    if x == 0:
        return 1
    else:
        return x * silnia(x-1)
```

Podstawowe typy danych Pythona:

- liczby
- typ logiczny (True, False)
- łańcuchy znaków
- listy
- zbiory
- krotki
- słowniki

Listy

- Listy są **uporządkowaną** kolekcją elementów, elementy umieszcza sie pomiędzy para nawiasów kwadratowych []
  `lst = [1, 2, 3]`
- Długość listy:
  `len(lst)`
- Nie musza zawierać elementów tego samego typu, mogą nawet zawierać listy zagnieżdżone:
  `lst2 = [1, "Kot”, 3.0]`
  `lst3 = [[1, "Kot”], 3.0]`
- Konkatenacja list
  `lst2 + lst3`

Zbiory (set)

- Zbiory są **nieuporządkowaną** kolekcja bez powtórzeń

- Przykład

```
print set("jego imie to Eryk i Eryk jest jego
  imieniem".split())
```

python 3

```
print(set("jego imie to Eryk i Eryk jest jego imieniem".split()))
```

Wynik działania:

"jego", "imie", "to", "Eryk", "i ", "jest", "imieniem"

Krotki (tuple)

- Krotki są niezmienną sekwencją elementów (nie da sie zmieniać wartości elementów składowych krotki po jej utworzeniu).
- Używane najczęściej w sytuacji, gdy nie powinno sie zmieniać elementów zawartych w kolekcji.
- Kolejne elementy krotki rozdziela sie przecinkami:
  `t = (12345, 6789, 'witaj!')`
- Krotki można zagnieżdżać:
  `u = t, (1, 2, 3, 4, 5)`

Krotki (2)

- Liczba elementów w krotce
  `len(t)`
- Odwołanie do elementów za pomocą indeksów, możliwe jest również robienie wycinków z krotek:
  `t[0]`
  `t[1:3]`
- Rozpakowanie krotki
  `x, y, z = t`

Słowniki (dict)

- Słowniki są kontenerem przechowującym pary klucz – wartość (**tablica asocjacyjna**). Kolejne rozdzielane przecinkami pary są zapisane pomiędzy nawiasami { }
```
telefony = {'Alek' :12345, 'Tadek' :11111}
```
- Odwołanie do wartości pod kluczem:
```
telefony = {'Alek']
```
- Przypisanie wartości do nieistniejącego klucza powoduje dodanie nowej pary:
```
telefony = {'Robert'] = 122222
```
- Przejście po elementach słownika:
```
for naz, nr in telefony.iteritems():
    print(naz, nr)
```

Słowniki (2)

- Wybór listy kluczy ze słownika:
  `telefony.keys()`
- Wybór listy wartosci ze słownika:
  `telefony.values()`
- Sprawdzenie czy klucz o podanej wartości istnieje w słowniku:
  `'Adam' in telefony`

Instrukcje

- Instrukcja przypisania
- Wywołanie funkcji
- Instrukcja print
- Instrukcja if/elsif/else
- Instrukcja for
- Instrukcja while/else
- Instrukcja pusta pass
- Instrukcja break
- Instrukcja continue
- Definicja funkcji def
- Szkielety

Instrukcje (2)

- Definicja klasy class
- Instrukcja powrotu z funkcji return
- Dostęp do modułu import
- Dostęp do składowych modułu from
- Wychwytywanie wyjątków try/except/finally
- Zgłaszanie wyjątków raise
- Dostęp do zmiennych globalnych/modułowych global
- Tworzenie generatorów yield
- Usuniecie nazwy del

Przykład

Klasy i obiekty

```
class Prostokat:
  zmienna = 12
  def __init__( self, a ):
      self.a = a
  def pole(self):
      return self.a * self.a

p = Prostokat(2)
print p.a
```

Przykład

Atrybuty

```
#atrybut klasowy
print Prostokat.zmienna
p.nowazmienna = 18
p.zmienna = 1000

#Odwołanie do zmiennej klasowej
print Prostokat.zmienna
```

Metody

- Metody mogą być wywoływane za pośrednictwem obiektu:
  `p.pole()`
- Można również przekazywać obiekt jako do metod klas jako pierwszy argument wywołania:
  `Prostokat.pole(p)`
- Metody specjalne – nazwa zaczyna sie i kończy od __
- Służą do celów specjalnych np.: inicjowania, definicji operatorów, konwersji typów, reprezentacji obiektów klasy w innej postaci (np. łańcuchów tekstowych)

Język Ruby

- Interpretowany, w pełni obiektowy i dynamicznie typowany język programowania
- Istnieje od 1995 r.
- Aktualna wersja stabilna: 2.3.2
- Oficjalna strona: http://www.ruby-lang.org

Ruby: Hello world

- plik hello_world.rb

```
#!/usr/bin/env ruby

# Hello world ruby program

puts 'Hello, world!'
```

Właściwości języka Ruby

- Ruby nie wymaga deklaracji zmiennych.
- Wszystko w języku Ruby jest obiektem.
- Typy podstawowe – obiekty zawierające pojedynczą wartość – jedną liczbę, jeden ciąg znaków, jedną datę.

```
wiek = 2
waga = 74.69
puts waga.round
puts 74.69.round
```

obiekt typu Fixnum

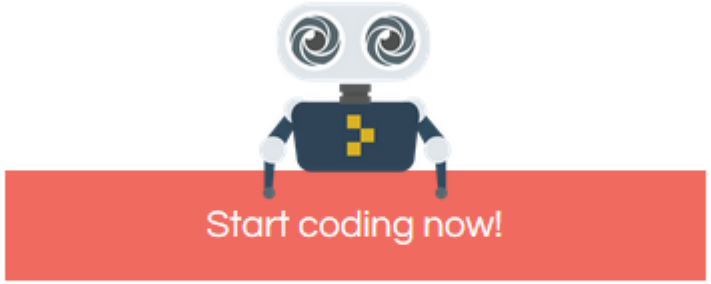
obiekt typu Float

Typy podstawowe, tablice

- Zmienne: `zm1 = 12, zMIENNA2 = 12.5`
- Stałe: `STALA1 = 10, Stala2 = 'b'`
- Łańcuchy znaków: `lancuch = 'Tekst'`
- Zmienne tablicowe: `liczby = [1, 2, 3],`
  `kwiaty = ['bez', 'mak', 'irys'],`
  `osoba = ['k', 60.5, 29, 'Ola']`
- Hasze czyli tablice asocjacyjne

Nauka podstaw języków

- Środowisko on-line – np.: repl.it
  - Python 3
  - Ruby
  - inne



FRAMEWORKI

Framework Django (Python)

- Wysokopoziomowy, opensource'owy framework przeznaczony do tworzenia aplikacji internetowych, napisany w Pythonie.
- Działa po stronie serwera.
- Opiera się na wzorcu projektowym podobnym do MVC nazywanym MVT (Model-View-Template).
- Pierwsze wydanie: 2005 r.
- Licencja: BSD.

Architektura MVT

- Model - opisuje strukturę danych, dotyczy schematu bazy danych
- View - kontroluje jakie dane mają dotrzeć do użytkownika
- Template - opisuje jak dane będą prezentowane
- Controler - dla Django nadzoruje parsowaniem URL

Framework Ruby on Rails (RoR)

- Opensource'owy framework do szybkiego tworzenia aplikacji webowych.
- Działa po stronie serwera.
- Pierwsze wydanie: 2004 r.
- Licencja: X11 (MIT).

Architektura Ruby on Rails

Model

- Klasy modeli
- Powiązania
- Walidacje
- Migracje

Biblioteka wspomagająca
ActiveRecord

ORM

Widok

- Szata graficzna
- Szablony i części
- Funkcje pomocnicze
- Zawartość statyczna

Biblioteka wspomagająca
ActionPack

Kontroler

- Klasy Kontrolerów
- Filtry
- Akcje

Biblioteka wspomagająca
ActionPack

Czy stosować frameworki?

- Duży czas potrzebny do zapoznanie się z Frameworkiem.
- Wykorzystując frameworki programista widzi masę niezrozumiałego kodu, który „coś” robi → znalezienie źródła błędu jest utrudnione.

Obiektowe Bazy Danych (OBD)

Obiektowe Systemy Zarządzania Bazami Danych (OSZBD)

OBIEKTOWY MODEL DANYCH

Wstęp historyczny do OBD

- **1970 r.** – badania nad OBD
- **1985 r.** – wprowadzono pojęcie „obiektowa baza danych”
- **1991 r.** – rozpoczęcie pierwszych prac nad standaryzacją
  - Utworzenie grupy Object Data(base) Management Group (ODMG). Grupa zaczęła działać formalnie jako organizacja; do 2001 r.
- **1999 r.** – rozpoczyna swoją prace Java Community Process, który w roku 2002 publikuje standard Java Data Objects 1.0, JSR243 – JDO 2.0, JDO 3.0
- **2005 r.** – Native Queries

Cel wprowadzenia OBD

- Obiektowe bazy danych wywodzą się z obiektowych języków programowania.
- Dają możliwość silniejszego powiązania z aplikacjami obiektowymi, niż było to w przypadku relacyjnych baz danych.
- Dzięki temu można zminimalizować ilość operacji związanych z przechowywaniem i dostępem do danych zorientowanych obiektowo.
- Obiektowe systemy zarządzania bazą danych zapewniają tradycyjną funkcjonalność baz danych, lecz bazują na modelu obiektowym

Standardy obiektowe

- Standard ODMG (1991)
- Model obiektu zaproponowany przez OMG
  - utworzenie standardu Common Object Request Broker Architecture (CORBA), 1991 r.
- Standard JDO (2002) - stworzony przez firmę Sun Microsystems
- Native Queries, Language INtegrated Query (LINQ), rozszerzenia: DLINQ, XLINQ
  - Używanie języków programowania do generowania zapytań

Standardy - ODMG

- Pierwszy standardowy model OBD
- Rdzeń modelu jako wspólny mianownik dla ORB (Object Request Broker), systemów bazodanowych, obiektowych języków programowania oraz innych aplikacji.
- Język specyfikacji obiektu ODL, od wersji 2.0 również inne języki – ułatwienie wymiany danych w rozproszonych bazach danych
- Obiektowy język zapytań OQL - język SQL wzbogacony o nowe możliwości.

SQL a OQL (ang. *Object Query Language*)

SQL	OQL
Typ rekordu →	Klasa
Krotka (rekord) →	Obiekt
Tabela →	Kolekcja obiektów
Schemat bazy danych →	Kolekcja klas

OQL - przykład

```
select b.biuroNr
from b in biura
where b.adres.miasto="londyn";
```

Standardy - JDO

- Jako podstawa użyty został model obiektu używany przez język Java.
- Język specyfikacji obiektu XML Metadata (pliki <klasa>.jdo)
- Obiektowy język zapytań JDOQL

JDOQL

- ang. *JDO Database Query Language*
- Przykład:

```
Query query =
persistenceManager.newQuery(Student.class,
  "age < 20");
Collection students =
  (Collection)query.execute();
```

Przykłady obiektowych baz danych

- GemStone – oparty na obiektowym języku Smalltalk, 1986 r.
- Cache
- Db4objects
- Inne: O_2, WakandaDB, ZODB, Versant, POET, Objectivity, ODI

http://db-engines.com/en/ranking

69.	↑ 75.	↑ 144.	InfluxDB	Time Series DBMS
70.	↓ 69.	↓ 65.	Caché	Object oriented DBMS
71.	71.	↓ 64.	OpenEdge	Relational DBMS
72.	72.	↑ 73.	Oracle NoSQL	Key-value store
73.	↑ 76.	↑ 78.	Amazon CloudSearch	Search engine
74.	↓ 73.	↑ 91.	Datameer	Document store
75.	↓ 74.	↑ 79.	Infinispan	Key-value store
76.	↑ 79.	↓ 69.	Teradata Aster	Relational DBMS
77.	77.	↓ 70.	Db4o	Object oriented DBMS
78.	↑ 87.	↓ 74.	EnterpriseDB	Relational DBMS

Dlaczego nie są popularne?

- Mało i słabo udokumentowane narzędzia.
- Stabilniejsze standardy dla baz relacyjnych.
- Bazy relacyjne są znane i udokumentowane.
- Małe lub brak wsparcia firm i organizacji.
- Język aplikacji zamiast SQL.
- ORM (np. Hibernate).

54

OBIEKTOWO-RELACYJNE SZBD

SQL3

- Standard języka SQL, opracowywany przez ISO oraz ANSI (*American Nat* ?), następca i rozszerz
- Uniwersalny język p cyjnych baz danych z eleme
- Specyfikacja SQL3 p :
  - zdefiniowanych prz nych typów danych (ADT),
  - włączając w to me ów, podtypy i dziedziczenie ora
  - integrację z języka
- Istniała koncepcja i i OQL według standardu C cyjne zostały zaniechane.

SQL	
Pojawienie się	1974
Paradygmat	wieloparadygmatowy
Typowanie	statyczne, typowanie silne
Implementacje	wiele
Pochodne	SQL-86, SQL-89, SQL-92, SQL:1999, SQL:2003, SQL:2006, SQL:2008
Aktualna wersja stabilna	SQL:2011/ 2011
Twórca	Donald D. Chamberlin i Raymond F. Boyce
Platforma systemowa	wieloplatformowy

Obiektowo-relacyjne SZBD

- PostgreSQL
- Oracle Database

PostgreSQL

- Producent: PostgreSQL Global Development Group
- Najnowsza wersja: 9.6.1
- Licencja: PostgreSQL
- Baz typu RSZBD (RDBMS) z rozszerzeniami obiektowymi

Oracle Database

- Producent: Oracle
- Najnowsza wersja: 12c
- Licencja: komercyjna
- Proceduralny język programowania PL/SQL
- Posiada rozszerzenia obiektowe – duża zgodność z SQL3

ORACLE®
DATABASE

Cechy obiektowe w SZBD Oracle

- SZBD Oracle implementuje znacznie większą część standardu SQL:1999 (SQL3).
- Abstrakcyjne typy danych
- Dziedziczenie typów
- Perspektywy obiektowe
- Tabele obiektowe
- Tabele zagnieżdżone
- Tablice o zmiennych rozmiarach
- Duże obiekty - LOB

Zalety OBD:

- Obiekty nie wymagają zmiany formatu oszczędzając czas zapisu i wykonania;
- Zredukowane stronicowanie;
- Łatwiejsza nawigacja;
- Lepsza kontrola współbieżności - hierarchia obiektów może być zablokowana;
- Model danych wzorowany na rzeczywistości;
- Działa dobrze dla rozproszonych architektur;
- Wymaga mniej kodu w językach obiektowych.

Wady OBD:

- Mniejsza wydajność dla prostych danych i powiązań;
- Tabele relacyjne są prostsze;
- Wiązanie późne może spowolnić dostęp;
- Więcej narzędzi dla baz relacyjnych;
- Standardy dla baz relacyjnych są stabilniejsze;
- Wsparcie dla baz relacyjnych jest pewniejsze.

Mapowanie obiektowo-relacyjne

ORM

Mapowanie obiektowo-relacyjne

- ORM (ang. *Object-Relational Mapping*) – sposób odwzorowania obiektowej architektury systemu informatycznego na bazę danych (lub inny element systemu) o relacyjnym charakterze.
- Implementacja ORM stosowana jest m.in. w przypadku, gdy tworzony system oparty jest na podejściu obiektowym, a system bazy danych operuje na relacjach.

Historia ORM-ów

1996 r. – pierwsze narzedzia do mapowania ORM dla Javy – TopLink.

1999 r. – powstaje J2EE – standard tworzenia wielowarstwowych aplikacji komponentowych.

2001 r. – pojawia się szereg bibliotek i narzędzi dla języka Java. W dziedzinie mapowania obiektowo-relacyjnego pojawiają się narzędzia Hibernate, iBatis. Największą popularność zyskuje Hibernate.

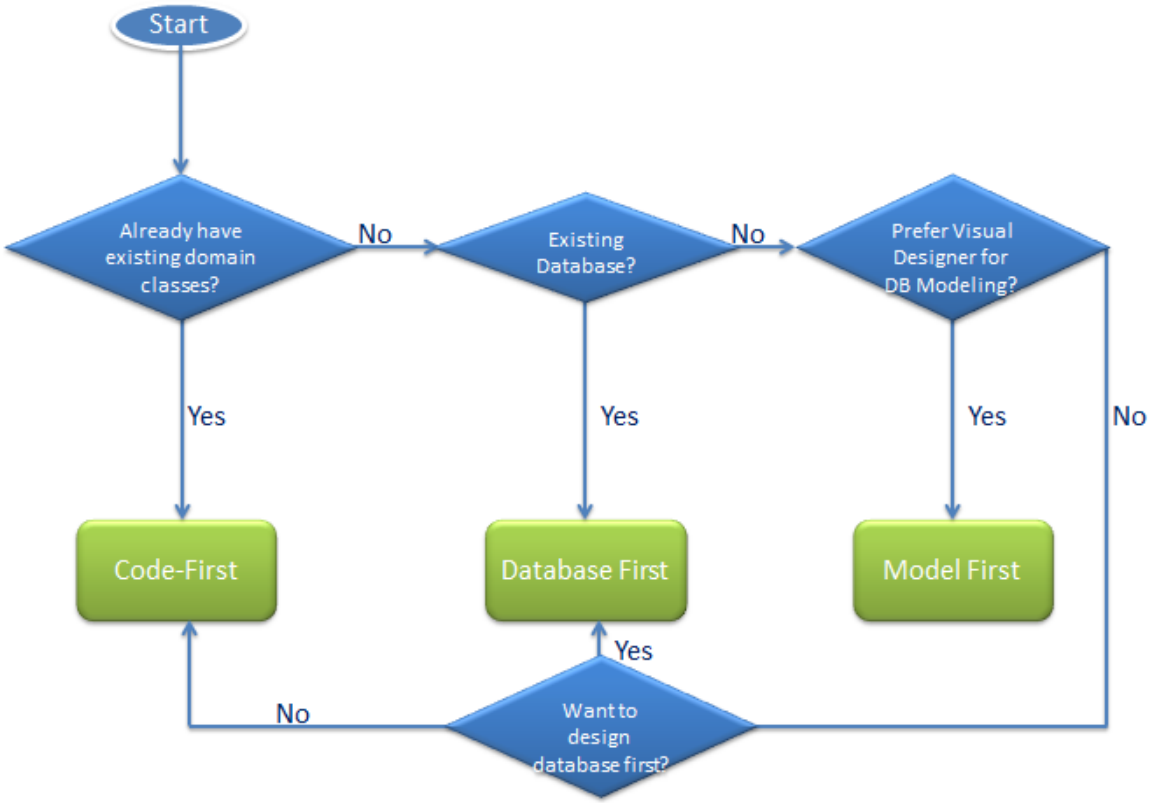
Kolejne lata – rozwiązania ORM trafiają do małych i średnich systemów informatycznych.

Przykładowe biblioteki ORM

- PHP: Doctrine, Propel, CakePHP
- Java: **Hibernate**, Object Relational Bridge, Torque, …
- C++: LiteSQL, ODB, Wt, QxOrm
- .NET: ADO.NET Entity Framework, NHibernate, LINQ
- Python: SQLAlchemy, ORM w Django, SQLAlchemy
- Ruby: ActiveRecord w RoR

Podejścia stosowane w ORM

- *Code-first*
- *Model-first*
- *Database-first*



Źródło: http://www.entityframeworktutorial.net/choosing-development-approach-with-entity-framework.aspx

Doctrine (PHP)

- ORM wzorowany na Hibernate i dostosowany do języka PHP
- Obiektowy język zapytań DQL (*Doctrine Query Language*)
- Pierwsze wydanie: 2008 r.
- Licencja: GNU Lesser General Public License
- Strona domowa: www.doctrine-project.org

Przykład

Doctrine - definicja klasy

```
class User
{
(...)

/**
* @var string
* @ORM\Column(name="name", type="string", length=255)
*/
private $name;

/**
* @var string
* @ORM\Column(name="email", type="string", length=255, unique=true)
*/
private $email;

(...)
}
```

Doctrine - dodawanie obiektu

```
$user = new User;
$user->setName("Kowalski");
$user->setEmail("jankowalski@poczta.pl");
$user->save();
echo 'Użytkownik o identyfikatorze '.$user
  ->getId().' został zapisany.';
```

Doctrine - związek *User-Book*

User:

```
/**
* @ORM\OneToMany(targetEntity=„Book", mappedBy=„owner”)
*/
private $books;


/**
* @ORM\ManyToMany(targetEntity="Group")
* @ORM\JoinTable(name="User_Group",
*      joinColumns={@JoinColumn(name="User_id",
   referencedColumnName="id")},
*      inverseJoinColumns={@JoinColumn(name="Group_id",
   referencedColumnName="id")})
*/
private $groups;
```

Doctrine - związek *User-Book*

Book:

```
/**
* @ORM\ManyToOne(targetEntity="user", inversedBy="books")
* @ORM\JoinColumn(name=„owner_id", referencedColumnName="id")
*/
private $owner;
```

Doctrine - dodanie związku

```
$book = new Book();
$book->setTitle("Pan Tadeusz");
$user->addBook($book);
$user->save();

$user->getBooks();

$book->getOwner();
```

Propel (PHP)

- Narzędzie ORM napisane w PHP
- Także integralna część frameworka Symfony
- Pierwsze wydanie: 2003 r.
- Licencja: open-source MIT license (licencja X11)
- Strona domowa: propelorm.org

Propel

Hibernate

- Framework do realizacji warstwy dostępu do danych.
- Zapewnia on przede wszystkim translację danych pomiędzy relacyjną bazą danych a światem obiektowym.
- Opiera się na wykorzystaniu opisu struktury danych za pomocą języka XML, dzięki czemu można „rzutować” obiekty na istniejące tabele bazy danych.

HIBERNATE

Hibernate (2)

- Licencja: GNU Lesser General Public License
- Strona domowa: hibernate.org
- Głównie dla Javy EE
- Używa języka zapytań HQL

Użycie Hibernate

```
SessionFactory sessionFactory = new Configuration().configure()
   .buildSessionFactory();
Session session = sessionFactory.openSession();
session.beginTransaction();
session.save(new Event("Pierwsze wydarzenie!", new Date()));
session.save(new Event("Kolejne!", new Date()));
session.getTransaction().commit();
session.close();

session = sessionFactory.openSession();
session.beginTransaction();
List result = session.createQuery("from Event").list();
session.getTransaction().commit();
session.close();

sessionFactory.close();
```

Konfiguracja Hibernate

```
<hibernate-configuration>
    <session-factory>
        <!-- Konfiguracja bazy danych -->
        <property  name="connection.driver_class">org.h2.Driver</property>
        <property name="connection.url">
            jdbc:h2:mem:db1;DB_CLOSE_DELAY=1;MVCC=TRUE
        </property>
        <property name="connection.username">sa</property>
        <property name="connection.password"/>
        <!-- SQL dialect -->
        <property name="dialect">
            org.hibernate.dialect.H2Dialect
        </property>
        <!-- Z XML -->
            <mapping resource="org/hibernate/tutorial/hbm/Event.hbm.xml"/>
        <!-- Z adnotacjami -->
        <mapping class="org.hibernate.tutorial.annotations.Event"/>
    </session-factory>
</hibernate-configuration>
```

Mapowanie za pomocą XML

```
<hibernate-mapping package="org.hibernate.tutorial.hbm">
   <class name="Event" table="EVENTS">
       <id name="id" column="EVENT_ID">
              <generator class="increment"/>
       </id>
       <property name="date" type="timestamp"
              column="EVENT_DATE"/>
       <property name="title"/>
   </class>
</hibernate-mapping>
```

Mapowanie za pomocą adnotacji

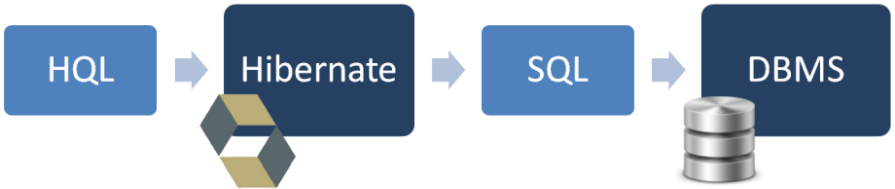
```
@Entity
@Table(name = "EVENTS")
public class Event {
   private Long id;
   private String title;
   private Date date;
   public Event() { }

   @Id
   @GeneratedValue(generator="increment")
   @GenericGenerator(name="increment", strategy = "increment")
   public Long getId() { return id; }
   @Temporal(TemporalType.TIMESTAMP)
   @Column(name = "EVENT_DATE")
   public Date getDate() { return date; }
   //Reszta getterów i setterów
```

Języki zapytań HQL, DQL

- Używa się odmian języka SQL, w których zamiast nazw tabel stosuje się nazw obiektów.
- Składnia zależna od użytej technologii.

Przykład (Pobranie obiektów klasy Klient, których pole wiek jest większe od 20):

```
SELECT u FROM User WHERE u.wiek > 20;
```



Zalety vs wady ORMów?

Zalety zastosowania ORMów

- Nie trzeba zajmować się problemami implementacji baz danych (zapytania SQL) – wystarczy skonfigurować ORMa
- Łatwość wytwarzania aplikacji
- Krótszy czas tworzenia aplikacji
- Duża przenaszalność aplikacji na różne systemy bazodanowe
- Zwiększenie bezpieczeństwa

Przenośność

- Systemy ORM, będące warstwą kodu wiążącą bazę danych z resztą aplikacji, oferują duże możliwości przenośności.
- Większość platform ORM obsługuje wiele baz danych.
- Podobnie jest w przypadku Django.
- Warstwa danych Django obsługuje PostegreSQL, MySQL, SQLite i Oracle.

Bezpieczeństwo

- We frameworkach wykorzystujących ORM (np. Django) bardzo rzadko wykonywane są przez programistów zapytania SQL → programy nie są podatne na wstrzykiwanie kodu (ang. *SQL injection*)
- Zastosowanie MVC - podział na warstwy daje mechanizmy zabezpieczania danych wejściowych (np. za pomocą inteligentnego cytowania znaków specjalnych)

Zasadnicze wady ORM

- W wypadku tworzenia aplikacji korzystających z gotowych baz danych mechanizmy synchronizacji ORMów nie wchodzą w rachubę.
- ORMy nie skłaniają do właściwego projektowania baz danych → spadek efektywności.
- Stosowanie ujednoliconych kluczy: pole autoincrement o nazwie 'id‘?

ORM - podsumowanie

- Umiejętność projektowania baz danych pozostaje jedną z kluczowych dla informatyka.
- Zastosowanie ORMów przyśpiesza wytwarzanie oprogramowania.
- ORM może być atrakcyjny jedynie w stosunkowo prostych projektach.

Dalsze lektury

- List of object-relational mapping software, https://en.wikipedia.org/wiki/List_of_object-relational_mapping_software

Pytania?

DZIĘKUJĘ ZA UWAGĘ