ORM - Intro

Spis treści

Wprowadzenie

Poniższa grafika zestawia zagadnienia, o których będziemy rozmawiać w ramach tego warsztatu.

Obraz 1. ORM → JPA → Hibernate

Zanim jednak do tego przejdziemy, przypomnijmy sobie bardzo krótko, o co chodziło w programowaniu obiektowym.

OOP (**Object-Oriented Programming**) - Podejście do definiowania programów za pomocą obiektów odzwierciedlających rzeczywistość. Podejście to jest związane z pojęciami: hermetyzacja, polimorfizm, abstrakcja i dziedzinie.

Gdybyśmy mieli w wielkim skrócie przedstawić programowanie obiektowe (lub zorientowane obiektowo) na jednej grafice, to mogłoby wyglądać to w ten sposób:

Obraz 2. OOP, Object-Oriented Programming

Natomiast definicja programowania obiektowego z Wikipedii mówi nam, że:

> (...) a programming paradigm based on the concept of "objects", which can contain data and code: data in the form of fields and code, in the form of procedures (...)

Co i jak się to ma do baz danych? Ten warsztat również będzie o tematyce baz danych, ale z innej strony. Dużo aplikacji na rynku korzysta z relacyjnych baz danych. Przy czym języki programowania zorientowane obiektowo (*ang. Object-Oriented Programming Languages*) nadal plasują się w czołówce najpopularniejszych. Widzieliśmy już na etapie poprzednich warsztatów poruszających zagadnienia baz danych, że mapowanie / łączenie / skojarzenie ze sobą obiektów w kodzie z tabelami w bazie danych jest nieodzownym elementem pracy z bazami danych. My dotychczas robiliśmy to w sposób "ręczny". Przyszła najwyższa pora, żeby poznać narzędzia, które mają nam w tym pomóc (i nie tylko).

ORM *(Object-Relational Mapping)*

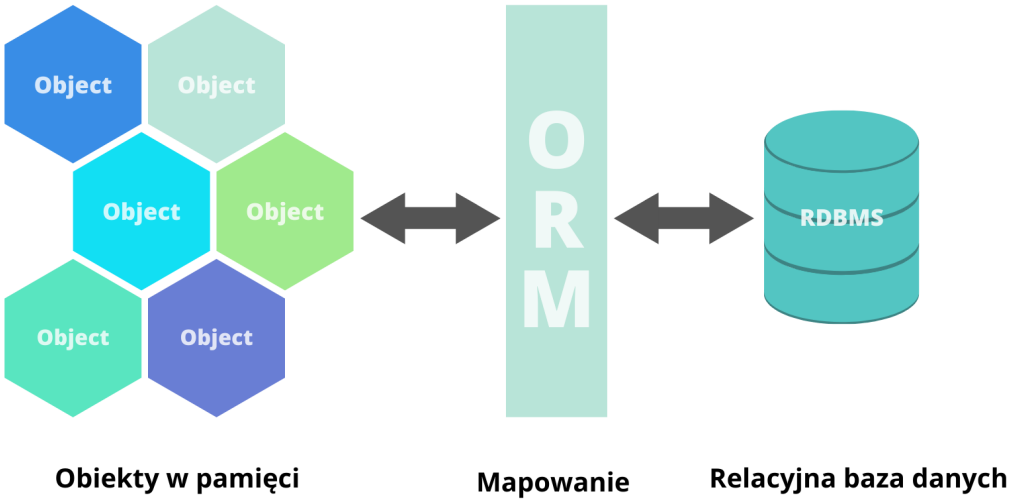
ORM *(Object-Relational Mapping)* - Mapowanie obiektowo relacyjne, czyli jak wpakować obiekt do relacyjnej bazy danych i z powrotem. Nie jest to takie łatwe zadanie, bo koncept obiektów i relacyjnych baz to dwa różne światy, które nie zawsze można odwzorować jeden do jednego. Pojęcie *ORM* nie dotyczy wyłącznie Javy, ale ogólnie obiektowego podejścia do programowania.

Definicja z Wikipedii wygląda tak:

> (...) a programming technique for converting data between incompatible type systems using object-oriented programming languages (...)

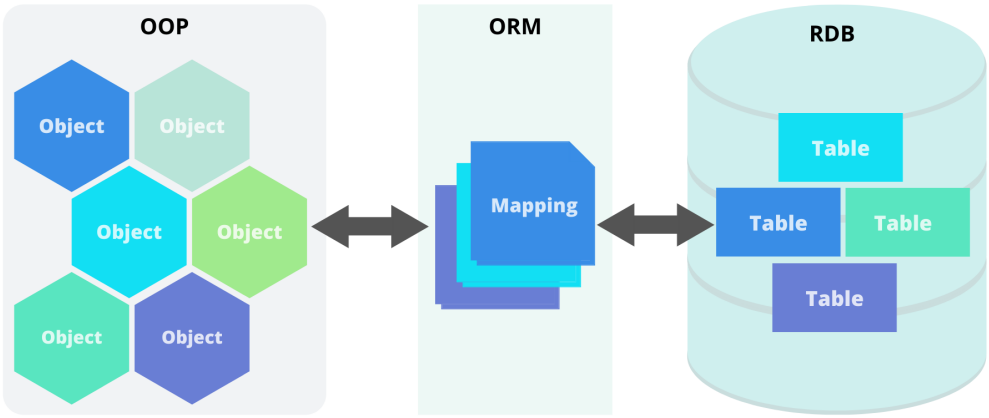
Poniższe dwie grafiki mają pomóc zobrazować do czego taki **ORM** jest potrzebny.

Schemat uproszczony:



Obraz 3. ORM, Object-Relational Mapping Basic

Schemat bardziej szczegółowy:



Obraz 4. ORM, Object-Relational Mapping Detailed

Zalety i wady ORM

- *Zalety*:
  - Zmniejsza ilość kodu i przyśpiesza proces tworzenia oprogramowania,
  - Pomaga spełniać zasadę **DRY** eliminując potrzebę powtarzania SQL,
  - Zwiększa bezpieczeństwo, utrudniając przeprowadzenie SQL Injection,
  - Pozwala na zmianę bazy danych bez dokonywania zmian w kodzie (w teorii),
  - Robi wiele za nas - np. wie jak przygotować SQL pod konkretną bazę danych
- *Wady*:
  - Gorsza wydajność dla skomplikowanych zapytań - wymaga pracy nad konkretnymi przypadkami, żeby tę wydajność poprawić,

- Robi wiele za nas - programista traci kontrolę nad tym, co się dzieje, co często prowadzi do trudnych do znalezienia błędów. Inaczej takie zjawisko w praktyce określa się słowem "magia".

W skrócie, używanie *ORM* się opłaca. Szczególnie że narzędzia *ORM* są na bieżąco rozwijane i są coraz lepsze.

Popularne narzędzia ORM

- *Java* - EclipseLink, TopLink, `Hibernate` - *to ten, o którym dalej będzie bardziej szczegółowo*,
- *.NET* - Entity Framework, NHibernate,
- *Python* - SQLAlchemy, Django ORM,
- *PHP* - Doctrine, Propel.

Na Wikipedii można też znaleźć tę listę, która przedstawia konkretne przykłady popularnego oprogramowania ORM z podziałem na języki programowania.

Encja

Wprowadźmy tutaj kolejne stwierdzenie jakim jest **Encja**. Stwierdzenie **encja** będzie używane w odniesieniu do **POJO**, które będzie używane do mapowania klas Java na bazę danych. Czyli inaczej mówiąc, jeżeli mówimy encja, to mamy na myśli klasę, która będzie używana w takim mapowaniu między Javą, a bazą danych. Jeszcze inaczej można powiedzieć, że encje to są takie specjalne **POJO**, które odzwierciedlają tabele w relacyjnej bazie danych.

ORM vs JDBC

A jak ma się **ORM** do **JDBC**? Przypomnijmy na początku czym było **JDBC**:

Java Database Connectivity, JDBC to API, które zapewnia nam podstawowe interfejsy, pozwalające na podłączenie się do bazy danych przy wykorzystaniu Javy.

Jak w ogólnym zestawieniu wyglądałoby natomiast porównanie **ORM** i **JDBC**? Chcę jednak zaznaczyć, że poniższe opisy są bardzo ogólnikowe, bo w wielu przypadkach można powiedzieć: *to zależy*.

ORM	JDBC
Działa wolniej	Działa szybciej
Dostarcza wygodniejsze sposoby definiowania zapytań	Wymaga zapytań SQL pisanych przez dewelopera
Sam mapuje klasy do encji bazodanowych	Mapowanie zostawia dla dewelopera
Zarządza połączeniem do bazy danych	Deweloper sam musi zarządzać połączeniem do bazy danych
Wspiera asocjacje (relacje) między tabelami	Nie wspiera asocjacji między tabelami

Poznaliśmy już sposób na pracę z bazami danych przy wykorzystaniu *JDBC* jak również *Spring + JDBC*, zaczynamy zatem wchodzić na *kolejny poziom abstrakcji*.