CZU: 371.3:004

# SOLUȚIONAREA DIFERENȚELOR DE PARADIGMĂ DINTRE MODELUL ORIENTAT PE OBIECTE ȘI MODELUL RELAȚIONAL PRIN INTERMEDIUL TEHNICII ORM

**Olga CERBU**, dr., conf.univ.
Universitatea de Stat din Moldova
**Gabriel TUREȚCHI**, student
Universitatea Tehnică a Moldovei

**Rezumat**. În cadrul acestei lucrări este abordată soluționarea diferențelor de paradigmă dintre modelul orientat pe obiecte (susținut de limbajele de programare de nivel înalt actuale) și modelul relațional (utilizat de cele mai populare sisteme de gestiune a bazelor de date). Soluționarea procesului automat de stocare a obiectelor într-o bază de date relațională folosind un framework ORM, constă în maparea obiectelor la tabelele corespunzătoare, asocierea dintre ele fiind descrisă folosind metadata. Pentru exemplificări sunt folosite limbajul Java și framework-ul ORM Hibernate, acesta beneficiind de o popularitate ridicată în rândul programatorilor.

**Abstract**. This article addresses the paradigm shifts between the object-oriented model (supported by current high-level programming languages) and the relational model (used by the most popular database management systems). Solving the automatic process of storing objects in a relational database using an ORM framework, consists in mapping the objects to the corresponding tables, the association between them being described using metadata. For example, the Java language and the Hibernate ORM framework are used, which enjoys a high popularity among programmers.

**Cuvinte cheie**: ORM, mapare, relație.
**Keywords**: ORM, mapping, relation.

## Introducere

Object / Relational Mapping (ORM) este o tehnică de programare ce face posibilă accesarea și manipularea obiectelor fără ca programatorii să fie interesați de sursa de date de unde provin aceste obiecte. Această tehnică a apărut din nevoia de a depăși diferențele de paradigmă dintre modelul orientat pe obiecte (susținut de limbajele de programare de nivel înalt actuale) și modelul relațional (utilizat de cele mai populare sisteme de gestiune a bazelor de date). Limbajele de programare orientate pe obiecte reprezintă datele într-un graf interconectat de obiecte, pe când bazele de date relaționale folosesc un mod tabelar de reprezentare. Efortul de a conecta atributele claselor definite prin intermediul unui limbaj orientat pe obiecte cu câmpurile tabelelor din baza de date nu poate fi ignorat, iar scopul unui ORM este acela de a crea o relație naturală, transparentă, fiabilă și de durată între cele două modele. Această nepotrivire de paradigmă pare să nu își fi găsit încă o soluționare definitivă care să fie aprobată de toți programatorii din industria IT, însă opinia generală este aceea că framework-urile ORM reprezintă un important pas înainte.

Procesul automat de stocare a obiectelor într-o bază de date relațională folosind un framework ORM, constă în maparea obiectelor la tabelele corespunzătoare, asocierea dintre

ele fiind descrisă folosind metadata. Un framework ORM complet include următoarele funcționalități:

- un API pentru operațiile CRUD (create, retrive, update, delete) aferente claselor persistente;
- un limbaj pentru specificarea interogărilor adresând clasele persistente și atributele acestora;
- un mod care să faciliteze definirea metadata pentru mapările dintre obiect și tabelă;
- o abordare consistentă a tranzacțiilor, a metodelor de stocare a datelor ("caching") și a asocierilor dintre clase;
- tehnici de optimizare în funcție de natura aplicației [1].

Acest articol propune o discuție de ansamblu asupra deciziei de utilizare a framework-urilor ORM prin descrierea câtorva dintre cele mai importante provocări și prin menționarea argumentelor pro și contra în aceste cazuri. Pentru exemplificări sunt folosite limbajul Java și framework-ul ORM Hibernate, acesta beneficiind de o popularitate înaltă în rândul programatorilor.

**Metode şi materiale aplicate**

API-ul de JDBC ("Java Database Connectivity") oferă acces universal la date din limbajul Java și este compus din două pachete: java.sql și javax.sql. Soluțiile existente pentru stocarea datelor folosind JDBC API sunt restrânse, se poate aminti interfața javax.sql.rowset.CachedRowSet reprezentând un container de înregistrări pe care și le stochează în memorie. Se constituie totodată o componentă JavaBeans ce dispune de scrolling și posibilități de actualizare și serializare. Implementarea de bază acceptă preluarea de date dintr-un set de rezultate, dar poate fi extinsă pentru a obține date și din alte surse tabelare. Cu un astfel de obiect se poate lucra în modul deconectat de baza de date, fiind nevoie de o conexiune doar atunci când datele trebuie sincronizate.

- Pe baza JDBC API s-au dezvoltat și alte biblioteci care oferă mai multe soluții de stocare a datelor în memorie, astfel de exemple fiind:
- Extensia Oracle pentru JDBC - oferă o interfață și implementare pentru statement cache (stocarea statement-urilor care sunt folosite în mod repetat)
- Implementarea PostgreSQL pentru JDBC - oferă un wrapper de stocare a statement-urilor peste implementarea de bază a JDBC-ului.

Există și alte soluții de stocare în memorie oferite de părți terțe care pot fi integrate într-o aplicație care folosește direct JDBC API. Astfel de exemple sunt:

- Commons JCS
- Ehcache

Din punct de vedere al framework-urilor ORM, acestea completează discuția despre "a nu discuta cu baza de date" cu subiectul "a nu discuta cu API-ul JDBC" în cazul Java, în sensul de a face transparentă programatorului relația cu API-ul JDBC, și de a manipula

obiectele stocate în memorie doar la nivelul ORM-ului și al implementărilor sale pentru astfel de stocări.

Hibernate oferă un prim nivel de stocare ("first-level cache") care e asociat cu sesiunea și e folosit implicit; mai oferă un al doilea nivel ("second-level cache)" care e asociat cu fabrica de sesiuni și e opțional (Figura 1).

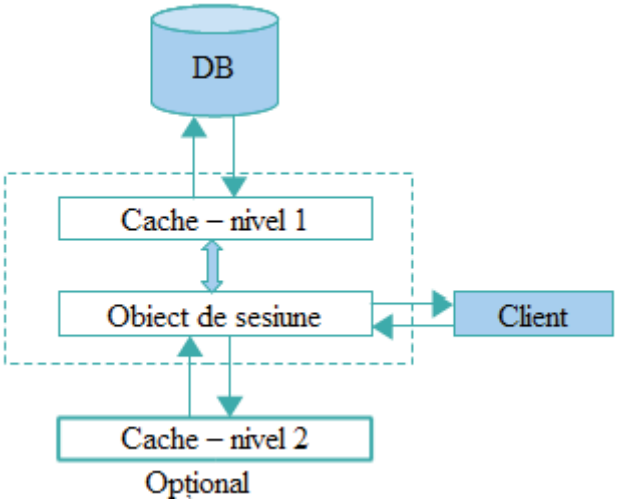


**Figura 1. Nivelele unu și doi de stocare a datelor oferit de Hibernate**

Primul nivel de stocare e folosit pentru a reduce numărul interogărilor pe baza de date la nivelul unei sesiuni [1]. Datele nu pot fi stocate în memorie și folosite de alte sesiuni în afara celeia care a adus datele inițial. Cel de-al doilea nivel de stocare este localizat la nivelul fabricii de sesiuni și este capabil de a depozita date din diferite sesiuni. Acest lucru înseamnă că toate obiectele de sesiune pot accesa aceleași date stocate. Hibernate suportă trei implementări "open source" pentru folosirea nivelului 2 de stocare. Acestea sunt:

- Ehcache
- OSCache
- JBoss TreeCache

Prin faptul că primul nivel de cache e folosit în mod implicit, Hibernate aduce deja un plus important în ceea ce privește performanța din punct de vedere al comunicării cu baza de date față de folosirea API-ul de JDBC. De asemenea, cel de-al doilea nivel de stocare, folosit în general pentru optimizarea aplicațiilor relativ bune din punct de vedere al performanței, duce la o creștere importantă a acesteia, iar programatorul are flexibilitatea de a alege implementarea care se potrivește cel mai bine contextului dat.

**Rezultate obținute**

În figurile de mai jos putem observa aplicarea ORM-ului Hibernate prin intermediul limbajului Java. Prin intermediul paradigmei POO a fost declarată o entitate “Files” cu atributele necesare (Figura 2).

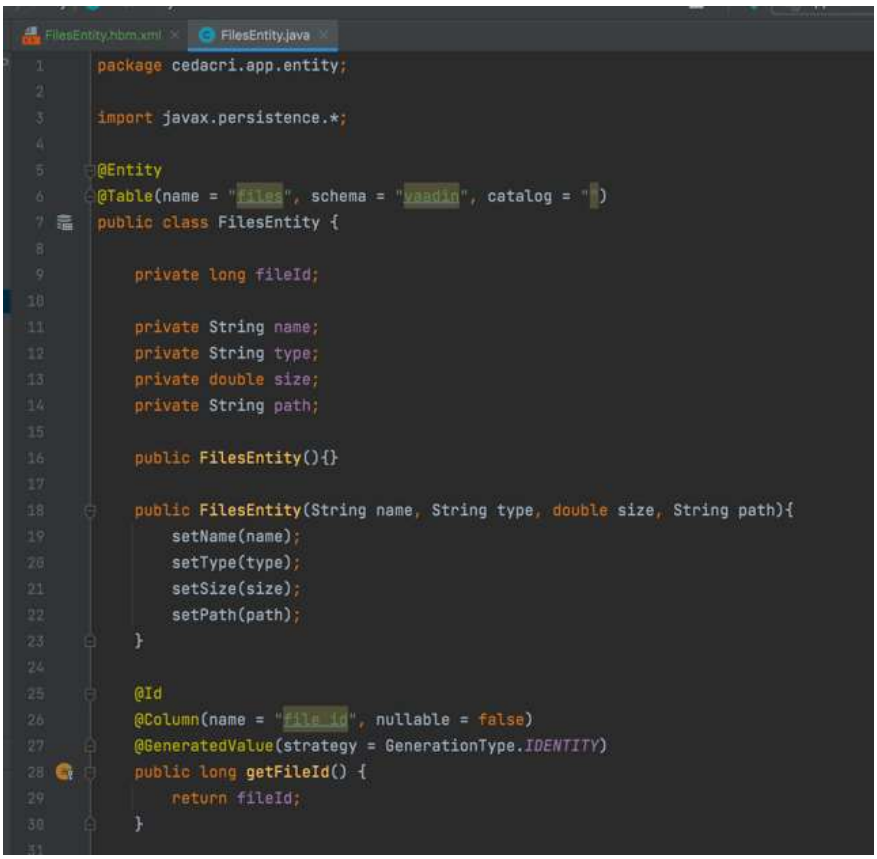


**Figura 2. Entitatea „Files” reprezentată ca clasă**

În fișierul XML (Figura 3) sunt mapate fiecare atribut a entității Files cu parametrii necesari pentru identificarea tipurilor de date care vor fi depistate de SQL.

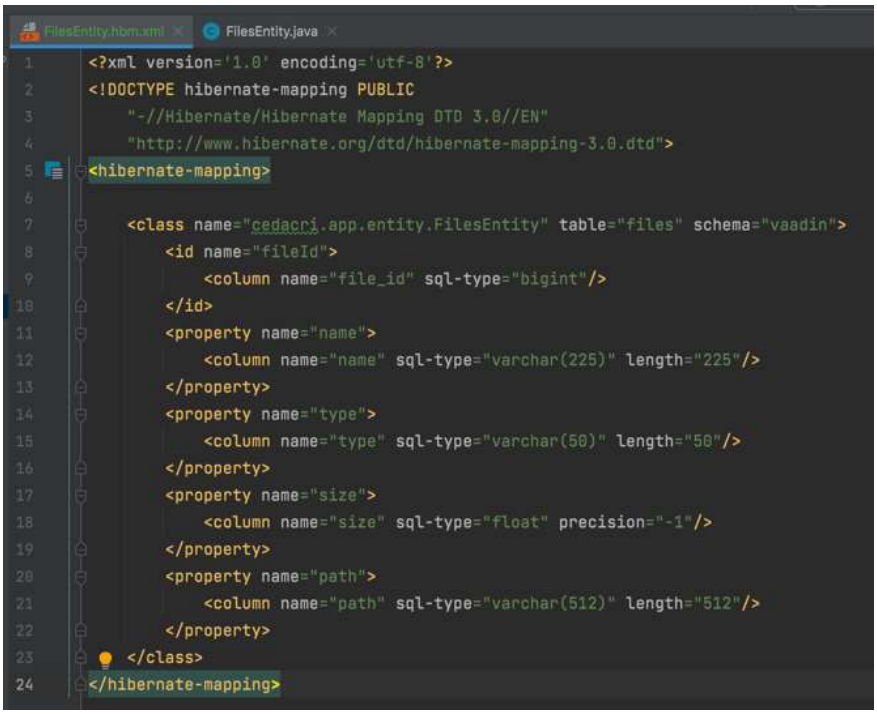


**Figura 3. Entitatea „Files” mapată în XML**

La executarea programului Java, Hibernate creează o conexiune la baza de date SQL și generează entitățile conform fișierului XML cu tipurile de date corespunzătoare (Fig. 4).


**Figura 4. Entitatea „Files” reprezentată în SQL**

## Concluzii

Folosirea unui framework ORM și alegerea lui sunt decizii care trebuie luate în cunoștință de cauză și care depind de specificul proiectului. Dificultatea inițială în dezvoltarea unui cod bazat pe un framework ORM constă în curba de învățare a framework-ului respectiv, urmată fiind de dificultatea mapării datelor la coloanele tabelelor, o operație care trebuie făcută manual ținând cont și de modul în care aceste relații vor defini în final structura tabelelor și a coloanelor. Curba de învățare ar trebui (cel puțin teoretic) să fie răsplătită pe termen lung, mai ales în cazul proiectelor medii și mari. Odată înțeles mecanismul ORM, timpul de dezvoltare ar urma să scadă ducând la o productivitate îmbunătățită a programatorului.

Pentru a folosi eficient un framework ORM nu ajunge experiența dobândită într-un limbaj de programare orientat pe obiecte, e necesară cunoașterea modelului relațional și a limbajului SQL. Framework-urile ORM permit evitarea scrierii de cod repetitiv și creșterea productivității, însă doar prin cunoașterea detaliată a framework-ului folosit el poate fi întrebuințat în mod optim, iar cunoașterea limbajului SQL ajută programatorul în cazul problemelor importante de performanță. Scopul final este acela de a avea productivitate și performanță în managementul datelor persistente.

## Bibliografie

1. Sourse: https://hibernate.org/orm/documentation/5.5/