HIBERNATE ORM ÎN JAVA

CEBAN Cristian

Universitatea Tehnică a Moldovei

Abstract: *În articolul dat sunt descrise caracteristicile principale ale Hibernate-ului, esența lui. Este descris procesul automat de stocare a obiectelor într-o bază de date relațională, folosind un framework ORM.*

Cuvinte cheie: *obiect-relațional, SQL, Java, Hibernate, ORM, session-per-request, session-per-conversation, model relational, mapare.*

1. Hibernate ORM

Hibernate ORM este un object-relational mapping framework pentru limbajul Java. Acesta oferă un framework pentru maparea a unui model de domeniu orientat pe obiect la o bază de date relațională. Hibernate rezolvă probleme de impedanță nepotrivire la obiect-relaționale prin înlocuirea bazei de date directă, baza de date persistenta accesează obiectele de nivel înalt cu funcții de manipulare.
Hibernate este un software gratuit care este distribuit sub licența GNU Lesser General Public
Prima versiune a fost realizată în anul 2001. Versiunea cea mai actuală la moment este Hibernate ORM 5.0.2.

Caracteristica principală a Hibernate-ului este maparea din clase Java la tabele de baze de date și maparea de la Java data types la SQL data types. Hibernate oferă, de asemenea interogarea de date și facilități de recuperare.

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, read, 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.

Componentele implicate în mecanismul de ORM sunt prezentate în figura 1.

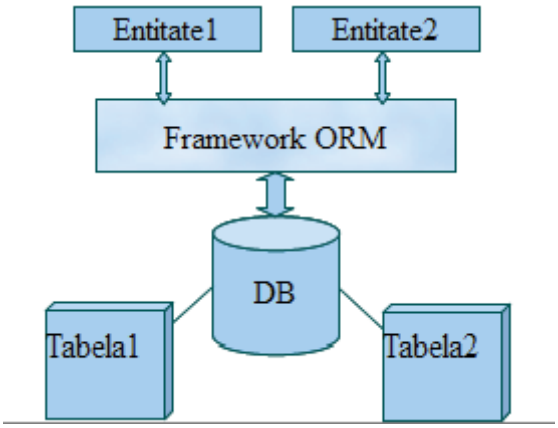

Fig. 1. Componentele implicate în mecanismul ORM. Cele două entități sunt persistate în tabelele corespunzătoare prin intermediul framework-ului ORM.

2. Provocări ale ORM. Concurență și tranzacții

Într-un mediu concurențial partajarea resurselor devine extrem de importantă și problema menținerii integrității datelor tinde sa devină complexă. Din acest motiv modul în care această partajare se implementează trebuie să fie cât mai simplu, să corespundă specificațiilor proiectului și să asigure integritatea datelor. În continuare vor fi descrise câteva dintre tehnicile de manipulare a datelor care trebuie folosite la un moment dat de programatori și care implică o atenție deosebită în ceea ce privește partajarea resurselor.

3. Blocarea optimistă a resurselor într-un mediu concurențial (optimistic concurrency)

Blocarea resurselor este folosită pentru ca datele să nu fie alterate între momentul citirii şi cel al folosirii. Varianta optimistă a acestei blocări se bazează pe presupunerea conform căreia mai multe tranzacții pot fi executate simultan fără ca una dintre ele să altereze datele alteia, în timp ce varianta pesimistă presupune blocarea unor resurse pe termen mai lung. Figura 2. descrie blocarea optimistă în varianta ei fundamentală.

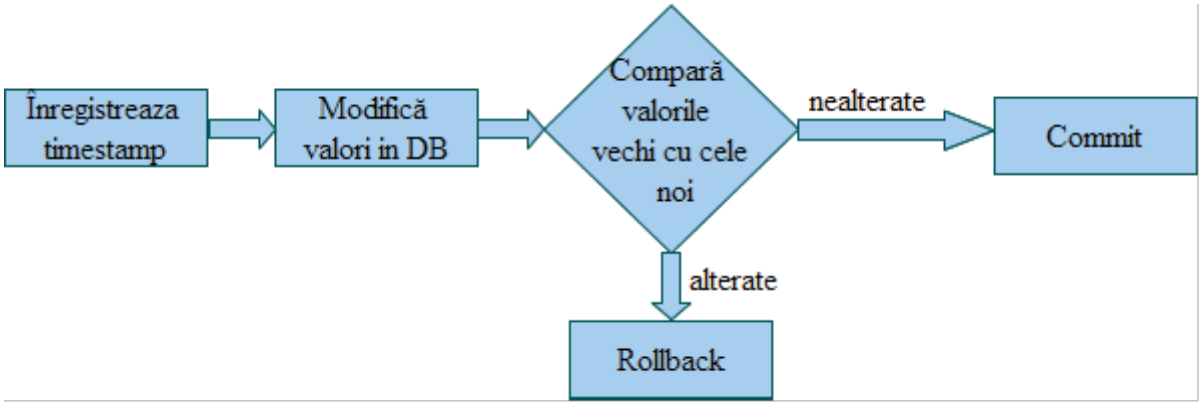

Fig. 2. Blocarea optimistă a resurselor

Într-o aplicație pentru care scalabilitatea şi concurența reprezintă cerințe non-funcționale importante se pune problema implementării concurenței cu "optimistic control", deoarece aceasta funcționează bine pentru scenarii cu citiri multiple şi modificări rare.

Din acest punct de vedere, într-o implementare care folosește JDBC API pur, fără ORM, responsabilitatea verificării integrității obiectelor încărcate şi care urmează a fi manipulate revine programatorului. În acest sens trebuie o verificare a versiunii obiectelor care poate fi făcută manual, însă doar în cazul scenariilor simpliste. Pentru scenariile complexe această sarcină devine dificilă întrucât se ajunge la grafuri de obiecte greu de urmărit şi controlat.

În această situație sunt dificil de implementat manual şabloane de tranzacții precum:

- sesiune per conversație ("session-per-conversation");
- sesiune per cerere ("session -per-request");
- sesiune per cerere cu obiecte detaşate ("session-per-request-with-detached-objects").

Pe de altă parte, folosirea unui ORM simplifică această sarcină. Spre exemplu, implementarea de Hibernate oferă implicit verificarea automată a versionării atât pentru sesiuni extinse cât şi pentru instanțe detaşate. Hibernate abordează problema concurenței cu optimistic control folosind versionare automată, obiecte detaşate şi sesiuni extinse.

4. Performanța interogărilor

Unul dintre cele mai puternice argumente în defavoarea ORM-urilor este cel conform căruia interogările SQL generate de instrumentele de mapare au o eficiență redusă în multe dintre cazuri în comparație cu interogările SQL scrise manual.

Pe parcursul dezvoltării aplicației se poate evita optimizarea prematură a interogărilor. Utilizarea unui framework ORM duce la scrierea mai rapida a codului, o lizibilitate mai bună, iar duplicările de cod sunt mai uşor de evitat. Odată cu creşterea numărului de interogări se poate folosi un instrument de verificare a vitezei de execuție pentru a identifica punctele sensibile ale performanței şi la diferite intervale de timp se poate interveni manual acolo unde e nevoie. De asemenea, printre cele mai la îndemână acțiuni care pot fi luate pentru optimizarea performanței e folosirea stocării în memorie a datelor şi indexarea bazei de date. Este de aşteptat ca performanța să fie acceptabilă, iar beneficiile folosirii unui mecanism ORM să depăşească problemele de performanță.

5. Maparea entităților

Performanța unui mecanism ORM şi productivitatea programatorului depind mult de o bună mapare a entităților la tabelele bazei de date. Printre riscurile unei mapări greşite se numără: generarea interogărilor neperformante;

- supraîncărcarea memoriei cu obiecte care nu sunt necesare în urma execuției interogărilor;

- productivitate sub aşteptări a programatorului din pricina scenariilor complexe determinate de mapări.

Folosirea eficientă a unei implementări ORM presupune o cunoaştere minimă a modului în care implementarea respectivă funcţionează "în interior" pentru a evita riscurile descrise mai sus.

6. Stocarea datelor în memorie

Stocarea datelor în memorie ("caching") reprezintă o modalitate de creştere a performanţei aplicaţiei, principalul său obiectiv fiind de a stoca anumite date în memorie, evitând astfel interogările pe baza de date pentru datele respective.

Concluzii

Platformele Java si .Net îşi dispută de ani supremaţia în domeniul dezvoltării aplicaţiilor software fie că este vorba de aplicaţii desktop, web sau mobile ambele oferă soluţii viabile dezvoltatorilor amatori sau profesionişti.

Tehnologiile de mapare relaţional obiectuală există de mulţi ani, Hibernate fiind printre primele poate chiar prima biblioteca ORM, realizată pentru platforma Java. Însă astăzi există atât NHibernate, reprezentând o portare a Hibernate pe platforma .Net, dar şi Entity Framework. S-a dovedit în nenumărate rânduri că atât Hibernate cât şi Entity Framework deţin capacităţile necesare implementării tehnicii ORM atât în abordarea *Model First* cât şi în abordarea *Database First.* Entity Framework creează clasele definite în modelul de date automat, Hibernate nu. Însă pe de alta parte din punct de vedere al dezvoltatorului crearea manuală a entităţilor şi a relaţiilor dintre ele ajuta mai mult la înţelegerea detaliilor unui program. Codul sursa generat dinamic de către Entity Framework poate avea dimensiuni mari, iar anumite atribute şi metode pot să nu fie folositoare, în timp ce scriere de mână conduce la obţinerea unui cod sursă ce conţine doar ce este necesar.

Se poate observa că pentru fiecare dintre cele două soluţii există avantaje şi dezavantaje, însă ceea ce este mai important este faptul că există soluţii multiple capabile să ofere toate facilităţile necesare dezvoltării aplicaţiilor. Astfel dezvoltatorilor nu le rămâne decât să se decidă care este varianta ce li se potriveşte cel mai bine.

Bibliografie

1. Oracle Corporation. Using Hibernate in a Java Swing Application. *netbeans.org*. [Interactiv] 2016. https://netbeans.org/kb/docs/java/hibernate-java-se.html#05.
2. Oracle Corporation. Connecting to a MySQL Database. netbeans.org/. [Interactiv] 2016. https://netbeans.org/kb/docs/ide/mysql.html.
3. V. Nick.Hibernate.javastudy.ru.[Interactiv]2015. http://javastudy.ru/hibernate/onetomany/.
4. V. Nick.. Hibernate — быстрый старт. Пример приложения Hibernate 5 Hello World. javastudy.ru. [Interactiv] 2015. http://javastudy.ru/hibernate/hibernate-quick-start/.
5. Seb Seby. Seby Seb Maparea Obiectual-Relaţională în Java Hibernate. prezi.com. [Interactiv] 2015. https://prezi.com/e7x8dhr0y4ur/maparea-obiectual-relationala-in-java-hibernate/.
6. Wikipedia. Hibernate (framework). wikipedia.org. [Interactiv] 2016. https://en.wikipedia.org/wiki/Hibernate_(framework).