ANALIZAREA MODELELELOR DE DATE OBIECT-RELAȚIONALE DJANGO ȘI DOCTRINE

Mihail ULINICI

Departamentul Ingineria Software și Automatică, grupa TI-191 F/R, Facultatea Calculatoare, Informatică și Microelectronică, Universitatea Tehnică a Moldovei

Autorul corespondent: Mihail ULINICI, e-mail: mihail.ulinici@isa.utm.md

Conducător științific: Dorian SARANCIUC, DISA, FCIM, UTM

Rezumat. *În articol dat este realizat o analiză coparativă a două modele de date: Django și Doctrine, acestea fiind modele de date obiect-relaționale. Este descris modul de serializare, mecanismele de protecție a integrității bazei de date incluse în aceste modele, și sisitemele de gestionare a bazelor de date pe care le suportă. Accentul fiind pus asupra paradigmelor folosite de aceste modele și de asemenea pe implementarea lor și anume: Active Record și Data Mapper, cu scopul de a reda cât mai clar diferența dintre aceste două modele, de asemenea punctând avantajele și dezavantajele la utilizarea unui model sau a altuia.*

Cuvinte cheie: *ORM, django, doctrine, data mapper, active record.*

Introducere

Mapare Obiect Relaţională (ORM) face posibilă citirea dar și manipularea obiectelor fară a ține cont de sursa de date de unde vin aceste obiectele. În principal a fost creat pentru a acoperi diferențele dintre modelul orientat pe obiecte și modelul relațional (folosit de majoritatea sistemelor de gestiune a bazelor de date). Tot o dată nu putem sa nu remarcam că astazi din ce in ce mai mult se atestă o tot mai mare tendință în direcția integrari cu SGBD (sisteme de gestiune a bazelor de date) non-relaţionale [1].

Redarea datele într-un graf interconectat este ceva specific limajelor de programare orientate pe obiect, iar bazele de date relaționale dupa cum știm folosesc o reprezntare tabelară. A conecta atributele clasi la câmpurile tabelelor bazei de date este indezirabilă, astfel scopul unui ORM este anume acela de a stabili o relație, transparentă și de durată între cele două modele.

Acțiunea de a stoca și salva obiecte într-o bază de date relațională folosind un framework ORM, implica transformarea obiectelor în tabelele corespunzătoare și mai apoi conexiunea dintre ele facânduse prin metadatele. Un framework ORM ne ofere funcționalități precum:[2]

- o interfață de programare (API) pentru a efectua operațiuni precum (create, read, update, delete), CRUD.
- un anume limbaj prin care ne-ar permite interogarea clasele și atributele acestora.
- un anume mecanizm care ne-ar permite definirea metadatelor pentru mapările dintre obiect și tabel.
- o tratare logica coerenta în gestionarea tranzacțiilor, a datelor care sunt stocate în cache și deasemenea asocierea între clasele respective.

1. Prezentarea ORM-urilor alese

1.1. Django ORM În domeniul tehnologilor web mai exact în libajul de programare Phyton putem spune ca Django este indispensabil. Acest limbaj are la baza concepte care previn evitarea și înlaturarea repetabilităţii, şi web-site-uri ghidate de baze de date. Astfel oferindune posibilitatea de a folosi interfeţe simple pentru operați de creare, citire, modificare şi ştergere a obiectelor. Django ORM include urmatoarele funcții importante:

- un sistem de a serializa şi valida datele din formulare HTML astfel încât acestea să poată fi salvate în baza de date,

- un sistem de serializare care poate genera şi citi reprezentări XML şi JSON, incât acestea sa fie în continuare citite și prelucrate,

- ofera o protecție ipotriva injecţiilor SQL și desemenea asupra atacurilor CSRF care au ca scop de prelua accesul asupra unui cont al unui utilizator deja conectat, interceptânt astfel date foarte importante uneori chiar cu caracter personal.

Django ORM poate fi utilizat cu o mulțime de sisteme de gestiune a bazelor de date precum: Oracle, PostgreSQL, SQLite, MySQL, Microsoft SQL Server. O variantă neoficială a acestuia este compatibilă deasemenea şi cu MongoDB [3].

Doctrine ORM are la baza mai multe biblioteci ale limbajului PHP care promit oferirea serviciilor de persistenţă şi a funcţionalităţilor asociate. Unul din lucrurile iportante este ca Doctrine folosește limbajului DQL, și asta ne permite foarte comod pentru a construi interogările necesare.

O interogare DQL poate fi:

$query = $em ->createQuery('SELECT u FROM MyProject\Model\User u WHERE u.age > 20').

Prin prezența un Query Bilder se face simplăma transforma din limbajul DQL în SQL. Însa nu permanent utilizam interogarile DQL, întrucât Doctrine are un set de funcţii ce ascund oarecum interogările parametrizate, făcând uneori neiportanta intelegerea pe deplin a limbajului SQL.

La bază, biblioteca Doctrine utilizeaza clasa PDO, facând ușor dar consistent accesarea bazelor de date. Putem observa asta în figura 1.

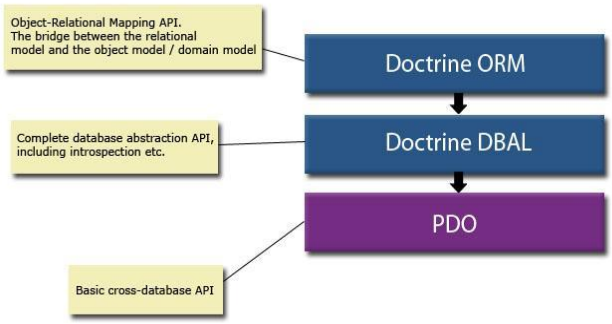

Figura 1. Structura bibliotecii Doctrine

Docrine ORM cuprinde o arie destul de mare fiind adaptabil cu multe SGBD relaţionale da nu numai, avem chiar și cu SGBD non-relaţionale cum ar fi : mongoDB, CouchDB, OrientDB, PHPCR.

În această analiză scurtă nu încerc sa compar ORM-urile alese din punc de vedere al limbajului în care au fost scrise. Ceea ce am încercat defapt este analiza din perspectiva paradigmelor implementate și mai exact, Active Record – de către Django ORM şi, Data Mapper de către Doctrine ORM.

2. Analiza comparativă a paradigmelor

2.1 Active Record. Acest model difera semnificativ de modelul Data Mapper în primul rând datorită perspectivei sale asupra obiectelor. Maparea obectelor se face exact pe un rând din tabel din baza de date, având aceleaș câmpuri și aceleaș coloane, iar în proiectele care folosesc acest ORM, nu poate fi vorba de o anumita indepentența fața de SGBD, însă un avantaj este ca ne permite dezvoltarea priectelor rapid și ușor. De asemena o altă trasatura importantă pe care o putem menționa este integrarea anumitor funcţiilor de interacţiune cu baza de date în componenta obiectelor. Un exeplu de acest fel pot fi conținerea funcțiilor delete() şi save().

În utilizarea este destul de usor de folosit insa nu respectă conceptul specializării claselor. Așa ca obiectele acestei clase nu au ca scop nu numai pastrarea informației denspre anumite procese dar și interacţionează cu baza de date.

2.2 Data Mapper

Conceptul de obiect în Data Mapper e puțin diferit fața de Active Record. Aici, obiectul este o înfățișare a business-proceselor și nu este nevoie ca acesta sa se mapeze exact pe un rând dintr-un tabel. Pot avea zeci de schimari între un rând din tabel și obiectul propriuzis. De asemenea, legătura cu baza de date nu se efectuează în constructorul clasei. În loc să fie incluse direct, informațiile despre proprietățile câmpurilor sunt stocate ca metadate sau așa zisele "doc-block-uri". [4]

Pentru efectuarea interogarilor aici lucrurile stau diferit, pentru acesta sunt folosite clase separate. Un exeplu ar fi pentru a edita un obiect se foloste un Entity Manager (Manager de entităţi).

Aceste Entităţile sunt obiecte cu identitate fiind practic la fel ca și obiectele din ORM-urile Active Record. Extragerea de date și deasemeane salvarea modificarilor asupra lor este și scopul un Manager de entităţi. Acesta este un exeplu de manager de entităţi :

```
$em->persist($user);
$em->flush();
```

Avem metoda perisist() care nu are nici un efect asupra bazei de date, ea doar înregistrează obiectul şi flush() care trimite spre execuţie interogarea.

Pattern-ul Data Mapper nu este foarte raspândit este folosit destul de rar în coparație cu Active Record care este foarte popular. Fapt ce se datorează în principal bibliotecilor care îl implementează deoarece sunt mai greu de folosit. Exemple de astfel de ORM-uri sunt Doctrine, Hibernate, SQL Alchemy, Nhibernate şi Slazure, DataMapper.

Concluzii

În alegerea ORM-ului potrivit, trebuie să ținem cont de mai mulți factori care ar trebui luați în considerare la fazele icipinte ale proiectului. Cu siguranța nu putem afirma că unul este mai bun ca altul, dar putem spune ca un ORM de tip Data Maper și anume Doctrine ar fi de preferat in realizarea proiectelor mari, unde regulile de bisness logica sunt explicite, de asemenea este mult mai usor de lucrat cu baze de date mari. Prin comparați Ative Recorder ar fi de preferat în proiectele simple, dezvoltarea și implimetare rapide, cu o imagine clar definita și înțeleasa asupra bazei de date. În final alegerea îi aparține specialistului care ar fi indicat sa prevada și situațile de dozvolatrea ulterioara a prioectului, astfel find conștient de problemele și rezolvarea lor.

Referințe

1. Martin Fowler, *Patterns of Enterprise Application Architecture, 2003.*
2. *Data Mapper Pattern* [online] [accesat 02.02.2023], Disponibil: https://en.wikipedia.org/wiki/Data_mapper_pattern
3. *Active Record Pattern*. [online] [accesat 02.02.2023], Disponibil: https://en.wikipedia.org/wiki/Active_record_pattern
4. *Diferenţa dintre Active Record şi Data Mapper* [online] [accesat 02.02.2023], Disponibil: http://culttt.com/2014/06/18/whats-difference-active-record-data-mapper/