UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Rok Logonder

UPORABA ORODIJ ZA OBJEKTNO-RELACIJSKO PRESLIKOVANJE PRI RAZVOJU JAVANSKIH APLIKACIJ

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

Mentor: viš. pred. dr. Damjan Vavpotič
Ljubljana, 2011

Univerza
v Ljubljani

Fakulteta za računalništvo
in informatiko

Tržaška 25
1000 Ljubljana, Slovenija
telefon: 01 476 84 11
faks: 01 426 46 47
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00086/2011
Datum: 01.04.2011

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **ROK LOGONDER**

Naslov: **UPORABA ORODIJ ZA OBJEKTNO-RELACIJSKO PRESLIKOVANJE PRI RAZVOJU JAVANSKIH APLIKACIJ**

USE OF OBJECT-RELATIONAL MAPPING TOOLS FOR DEVELOPMENT OF JAVA BASED APPLICATIONS

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Preučite in predstavite uporabo orodij za objektno-relacijsko (O-R) preslikovnje pri razvoju javanskih aplikacij na primeru konkretnega orodja. V nalogi se posvetite tako teoretičnim kot praktičnim vidikom uporabe O-R preslikovalnikov ter predstavite njihove prednosti in slabosti. Predstavite tudi kaj prinaša uporaba O-R preslikovalnikom podjetjem, ki se ukvarjajo z razvojem informacijskih sistemov.

Mentor:

viš. pred. dr. Damjan Vavpotič

Dekan:

prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

Zahvala

Zahvalil bi se rad svoji družini za omogočanje študija ter podporo pri le-tem, svojim sošolcem, ki so mi olajšali sam študij na fakulteti ter vsem, ki so v času študija tako ali drugače vplivali name kot na študenta in kot na človeka, predvsem dekletu Jerneji.

Posebna zahvala gre mentorju, viš. pred. dr. Damjanu Vavpotiču, ki me je s hitrimi odzivi na moja vprašanja, nasveti in predlogi vodil k dokončanju diplomske naloge.

Zahvaljujem, se tudi podjetju ZZI d.o.o. za vse, kar so mi v preteklih štirih letih omogočili.

Kazalo vsebine

Kazalo slik

Kazalo tabel

Povzetek

Pri razvoju javanskih aplikacij, se prej ko slej srečamo s shranjevanjem podatkov. Navadno za shranjevanje podatkov uporabimo relacijsko podatkovno bazo. Za sam dostop do podatkovne baze se moramo povezati s strežnikom, na katerem je podatkovna baza. Za povezovanje s podatkovno bazo ter zagotavljanje trajnosti objektom, pa obstajajo objektno-relacijski preslikovalniki. Ta orodja nam vzpostavijo privid, da tabele iz relacijske podatkovne baze, programer vidi kot običajne javanske razrede, njihovi atributi pa so lahko trajni. Poznamo več vrst objektno-relacijskih preslikovalnih orodij. Poznamo take, ki preslikajo rezultate klice poizvedb na podatkovni bazi ter objektno-relacijske preslikovalnike, ki se držijo javanskih standardov za zagotavljanje trajnosti podaktov. Aplikaciji taka orodja skrajšajo razvojni čas ter večajo preglednost kode. Uporaba teh orodij postaja vedno bolj priljubljena, razvoj objektno-preslikovalnih orodij napreduje pa iz dneva v dan. V določenih situacijah se odločimo za objektno preslikovalno orodje, ki je takrat najprimernejše. Pri razvoju aplikacije, kjer ne bo veliko hkratnih dostopov do istih podatkov v podatkovni bazi, potrebujemo objektno-relacijsko orodje, ki bo predvsem enostavno za uporabo, koda pa pregledna. Pri razvoju aplikacije, kjer je hkratnih dostopov do istih podatkov na podatkovni bazi več, se odločimo za objektno-preslikovalno orodje, ki potrebuje ob inicializaciji in klicih poizvedb na podatkovno bazi čimmanj časa ter zasede čimmanj prostora na pomnilniku. V diplomski nalogi sem primerjal objektno-preslikovalna orodja MyBatis, Hibernate ter DataNucleus. V primeru, da razvijamo aplikacije, kjer ne pričakujemo velikega števila hkratnih povpraševanj po istih podatkih v podatkovnih bazah bi se, izmed naštetih, za uporabo odločili za eno izmed orodij Hibernate in DataNucleus. Pri razvoju tehnično zahtevnejših javanskih aplikacij, pa bi se odločili za orodje MyBatis.

Ključne besede: objektno-relacijsko preslikovanje, trajnost, MyBatis, Hibernate, DataNucleus.

Abstract

Whilst developing Java applications, we sooner than later come upon the data storing problem. Usually for saving data, we use a relationa database. For accesing the database we must connect to the database server. The means of connecting to the database and guaranteeing persistence to objects are provided by tools called object relational mappers. These tools provide the ilusion of dealing with tables in a relational database the same way as if the developer was dealing with Java objects. There are many kinds ob object-relational mapping tools on the market. For example, there are Object-Relational mapping tools that just map the results of queries into simple java objects and those, which use Java standards to ensure persistency of data. Use of these kind of tools shortens the developement time and decrease the size of workable code. The use of these kind of tools is becoming more popular each day so object relational mapping tools are being developed all the time. In certain situations we decide for the most appropriate data mapping tool. Whilst developing applications, where not many demand for data will be simultaneous, we normally decide for a data mapping tool which is simple to use and ensures viewable code. On the other hand, whils developing application where simultaneous access to the same data on the database are many, we decide for a data mapping tool that needs, at initialisation time and at executing query time as little time as possible. In this work, i compared object relational tools MyBatis, Hibernate and DataNucleus. Depending on the type of application i would decide for Hibernate and DataNucleus while developing less demanding application where simultaneus calls for the sam data are not many, and MyBatis for developing more advanced Java applications.

Key words: object reational mapping, persistency, MyBatis, Hibernate, DataNucleus

1 Uvod

Pri razvoju javanskih aplikacij se pogosto srečamo s težavo, kako se povezati z bazo, da nam ne bo vzelo veliko časa, hkrati pa bo uporabljena metoda učinkovita. Z uporabo vmesnikov knjižnice java.sql, kot na primer Statement, Connection, ResultSet, itd. si, ko pride do večih klicev na podatkovno bazo znotraj ene aplikacije ali pa postanejo poizvedbe na podatkovno bazo kompleksnejše, se nam tudi sam čas razvoja aplikacij poveča. V takih primerih je smiselno uporabiti orodja, ki uporabljajo objektno-relacijsko preslikovanje. Orodja, ki nam omogočajo objektno-relacijsko preslikovanje nam tabele v podatkovni bazi, ali pa rezultate poizvedb, na podatkovno bazo, prikažejo kot javanske objekte in s tem pri razvoju zahtevnejših aplikacij zmanjšajo količino programske kode, razbremenijo nas večino dela glede pretvarjanja tipov podatkov iz podatkovne baze v javansko aplikacijo itd. Orodja, kot so Hibernate, DataNucleus in MyBatis, nam s tem omogočajo večjo preglednost razvite programske kode, meja med delom programske kode, ki skrbi za dostop do podatkovne baze, se s konfiguracijskimi in preslikovalnim datotekami še bolj loči od programske kode, ki skrbi za samo logiko v aplikaciji, tako da lahko npr. razvijalec, zadolžen za razvoj logike v aplikaciji resnično upravlja le z logiko aplikacije – vse rezultate poizvedb na podatkovni bazi vidi kot osnovne javanske objekte, do želenih podatkov pride z enostavnim klicem get in set metod.

Orodja za O-R preslikovanje se med seboj razlikujejo po načinu delovanja. V okviru naloge sem predstavil dva tipa orodij:

- orodja, ki preslikajo rezultate poizvedb v objekte, kot npr. MyBatis in
- orodja, ki preslikajo tabele iz relacijske podatkovne baze, ter njihove relacije v javanske objekte in obratno: iz javanskih objektov v tabele v podatkovni bazi. Ta orodja se razlikujejo tudi po standardih, ki jih uporabljajo – JDO in JPA.

Vsako izmed obravnavanih orodij nam zagotavlja trajnost želenih podatkov.

1.1 Cilj diplomske naloge

Kot cilj diplomske naloge sem si zadal, da ugotovim, kako delujejo trajnostna objektno-preslikovalna orodja in pa katero orodje je najbolj smiselno uporabljivo pri razvoju določenega tipa javanskih aplikacij glede na kriterije:

- Čas razvoja kode za povezavo s podatkovno bazo.
- Število vrstic kode in število datotek, ki jih uporabimo za delovanje.
- Hitrost pridobivanja podatkov iz poizvedb na podatkovni bazi.

2 Opis teorije

2.1 Osnovni pojmi

2.1.1 Relacijske podatkovne baze

Relacijske podatkovne baze so zbirka podatkov, urejena kot niz formalno opisanih tabel, iz katerih se lahko do teh podatkov dostopa. Izumitelj relacijskih podatkovnih baz je E.F. Codd iz podjetja IBM leta 1970. Standardni uporabniški in aplikacijski programski vmesnik do relacijske podatkovne baze je strukturirani poizvedbeni jezik – SQL. Relacijska podatkovna baza je pravzaprav niz tabel, ki vsebujejo podatke, te tabele pa so lahko odvisne druga od druge.

2.1.2 Java

Java je visokonivojski programski jezik, razvit iz strani Sun Microsystems. Java je objektno orientiran programski jezik, v sintaksi podoben jeziku C++. Javanska osnovna koda je shranjena v datotekah s končnico .java in se prevaja v obliko imenovano bytecode – v datoteke s končnice .class. To prevedeno obliko lahko požene Javin interpreter. Prevedena javanska koda lahko teče na večini računalnikov, to pa je zato, ker javini intepreterji in pogonska okolja, poznana kot Java virtualni stroji (JVM), obstajajo za večino operacijskih sistemov.

2.1.2.1 Objekt

Java jo objektno naravnan programski jezik, kar pomeni, da so objekti osnovni gradniki vsake javanske aplikacije. Objekt je ena instanca razreda, razred je model, po katerem se objekti kreirajo. Imamo lahko model avtomobila, ki vsebuje atribute vrata, kolesa, motor, objekt pa kot instanca razreda predstavlja avto z npr. zelenimi vrati, vdrtimi kolesi in BMWjevim motorjem. **[12]**

2.1.3 Trajnost

Trajnost podatkov je stanje podatkov, ki zagotavlja, da so podatki dostopni tudi preko življenjskega cikla v aplikaciji, podatki pa se shranijo na shranjevalni enoti (trdi disk...). Za objektno naravnane programske jezike, kot je Java, trajnost zagotavlja, da so podatki dosegljivi tudi potem, ko se je metoda, ki jih je pridobila iz podatkovne baze že nehala izvajati. Podatki postanejo trajni, objektno preslikovalna orodja uporabljajo trajnost za shranjevanje pridobljenih podatkov v objekte, atribute,..

Brez trajnosti podatkov bi stanje atributov obstajalo le v pomnilniku in bi se izgubilo ob prvi nepravilnosti v delovanju pomnilnika (izklop računalnika iz električnega omrežja...).

2.1.4 Refleksija

Refleksija je proces, s katerim lahko aplikacija spremeni lastno strukturo in obnašanje ko je že prevedena in se poganja v javinem navideznem stroju. Omogoča vpogled v razrede, vmesnika, polja in metode, brez da bi poznali imena vmesnikov, metod in polj v času prevajanja. Prav tako omogoča izdelavo novih instanc objektov in klice metod. **[10]**

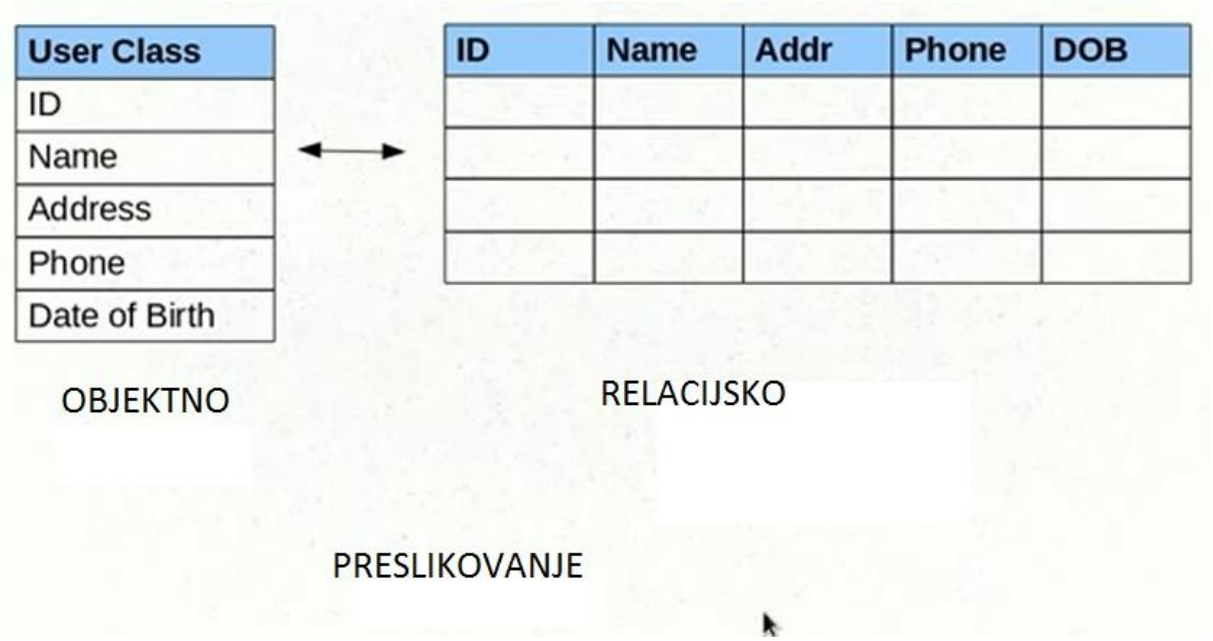
2.1.5 Razširjanje

Razširjanje je proces, ki prevedeni javanski kodi doda bite, ki razširijo funkcionalnost prevedene kode. V primeru preslikovalnih orodij, ki se držijo standarda JDO, se razširjanje uporablja zato, da se modelnim razredi ter njihovim atributom dodeli lastnost trajnosti. **[16]**

2.2 Objektno–relacijsko preslikovanje

2.2.1 Zakaj se uporablja objektno–relacijsko preslikovanje

Objektno-relacijsko preslikovanje (ORM) je koncept, ki razvijalcem v objektno orientiranih okoljih omogoča, da v svojih aplikacijah upravljajo s podatki na povsem domač način - kar preko objektov tako, kot da bi delali z objektno podatkovno bazo in ne z relacijsko. To omogoča abstrakcijski sloj ORM, ki poskrbi za transparentno preslikovanje iz okolja objektov v relacijsko podatkovno bazo in nazaj.



Slika 1: Objektno-relacijsko preslikovanje..

Ideja objektno-relacijskega preslikovanja se je pojavila še pred večjim razmahom objektno orientiranega programiranja v 90-ih letih, a kljub temu je še dolgo ostala le v domeni akademskih projektov in redkih komercialnih enterprise sistemov. V zadnjih letih se je predvsem po zaslugi nekaterih odprtokodnih projektov ORM razširil praktično na vsako platformo, danes pa je tudi integralen del vsakega sodobnega ogrodja za razvoj spletnih aplikacij.

2.2.2 Standardi

Pri objektno-preslikovalnih orodjih se srečamo z dvema standardoma za preslikovanje – JDO in JPA.

2.2.2.1 JDO

JDO – Java Data Objects - je standardni način dostopanja do trajnostnih podatkov v podatkovnih bazah z uporabo navadnih Javinih objektov za prikaz željenih podatkov. Pristop loči upravljanje s podatki – s klicem javanskih funkcij in pa upravljanje s podatki na nivoju podatkovne baze.

Visokonivojski JDO API je prilagojen, da zagotovi transparenten vmesnik za raziskovalce, za shranjevanje podatkov, brez da bi se jim bilo treba naučiti novega jezika za dostop do podatkovne baze za vsak tip podatkovnih baz. JDO lahko implementiramo z uporabo nizkonivojskega API-ja (kot je JDBC) za shranjevanje podatkov. Razvijalcem omogoči pisanje javanske kode, ki transparentno dostopa do želene podatkovne baze, brez da bi uporabljali kodo, značilno za dostop do baze. JDO je bil razvit kot JSR v Java Community Process programu: prva različica je bila JDO 1.0, trenutno pa se razvija različica pod šifro JSR 243 - 2.1.

JDO 2.0 je popolnoma podprta JSR specifikacija za trajnost povsem navadni javanskih objektov (POJO). Obstaja več orodij, ki se drži JDO standardov, npr. DataNucleus.

JDO v procesu postavitve uporablja »razredno razširitev«. Vsak trajen razred mora biti razširjen, da je lahko uporabljen iz strani JDO okolja v času delovanja. Trajni razredi so prevedeni z uporabo tradicionalnega javanskega prevajalnika, da dobimo datoteke tipa .class. Razširjevalnik prebere te .class datoteke in JDO metapodatke in naredi razširjene .class datoteke. Te razširjene datoteke so neodvisne od tipa podatkovnega zbira. Proces razširjanja ne spremeni funkcijskih lastnosti razredov, ki jih razširjamo. Doda jim kodo, da zagotovi, da se lahko vse vrednosti polj preberejo iz podatkovnega zbira in da so spremembe teh vrednosti zabeležene. **[4] [5] [9]**

2.2.2.2 JPA

JPA – Java Persistence API – je standardni API, ki se uporablja za urejanje trajnosti podatkov in objektno–relacijskega preslikovanja. JPA je dodan v Java EE 5 platformi. Vsak aplikacijski strežnik, kompatibilen z Java EE5, podpira JPA. Sestavljajo ga naslednji deli:

- Java trajnostni API
- Objektno – preslikovalni metapodatki
- Poizvedbeni jezik

JPA je zgrajen na podlagi najboljših idej iz trajnostnih tehnologij, kot so TopLink, JDO in Hibernate. JPA je sedaj obravnavan kot standardni komercialni pristop k objektno–relacijskemu preslikovanju v razvoju javanskih aplikacij. JPA sam po sebi je le specifikacija, ne produkt; ne zna zagotavljati trajnosti ali česarkoli podobnega. JPA je le niz vmesnikov in potrebuje implementacijo. Obstajajo odprtokodne in komercialne implementacije. JPA lahko preslika le objekte z relacijsko podatkovno bazo, z ostalimi tipi podatkovnih baz preslikava ni mogoča. Zaradi izredno velike ponudbe na področju relacijskih podatkovnih baz – Oracle, DB2... je bil sam razvoj močno pospešen.

JPA zagotovi običajnim javanskim razredom trajnost. Omogoča, da se piše nastavitve za objektno - relacijsko preslikovanje kot standardne anotacije ali pa XML datoteke. JPA definira tudi EntityManager API (on runtime) za procesiranje poizvedb in transakcij za objekte z podatkovno bazo.

Orodja, ki se držijo JPA smernic za samo preslikovanje objektov z relacijsko podatkovno bazo uporabljajo tako refleksijo, kot razširjanje prevajane kode. **[13]**

2.2.3 Primer delovanja objektno-relacijsko preslikovalnih orodij, ki se držijo standardov JDO ali pa JPA

Vzemimo primer razreda Uporabnik. Uporabnik ima običajna atribute – ime, geslo... Uporabljamo 5 instanc razreda uporabnik. Če želimo podatke posameznih uporabnikov shraniti, ponavadi uporabimo podatkovno bazo, tako da morajo biti objekti, v katere preslikujemo, trajni. V podatkovni bazi potrebujemo korespondenčno tabelo, v tem primeru

imamo v podatkovni bazo tabelo Uporabnik s stolpci, korespondenčnimi z atributi v objektu Uporabnik. Torej: razred Uporabnik je korespondenčen tabeli v podatkovni bazi, instanca razreda uporabnik pa je korespondenčna vrstici v tabeli uporabnik v podatkovni bazi.

3 Objektno–relacijski preslikovalniki

3.1 Seznam in primerjava preslikovalnikov

V diplomski nalogi so predstavljeni trije preslikovalniki: MyBatis, Hibernate in DataNucleus. Ti trije preslikovalniki so bili izbrani, ker vsak predstavlja drugačen tip objektno relacijskega preslikovalnika. Vsi so brezplačni:

- MyBatis ni tipičen objektno-relacijski preslikovalnik – za svoje delovanje ne uporablja standarda JDO ali JPA, pač pa ima svoj skelet programja ter preslikuje zgolj rezultate poizvedb, ki jih napišemo sami

-Hibernate je objektno–relacijski preslikovalnik, ki se pri svojem delovanju drži standarda JPA, za zagotavljanje trajnosti podatkom pa uporablja refleksijo.

-DataNucleus je objektno-relacijski preslikovalnik, ki se pri svojem delovanju drži standarda JDO.

Pri primerjavi sem si zadal tri kriterije primerjave. Primerjal sem:

- Število vrstic kode in datotek, da sem vzpostavil povezavo do podatkovne baze ter povezovalni čas do podatkovne baze
- Enostavnost uporabe – nivo zahtevnosti ter podpora na spletu.
- Hitrost delovanja enostavnih poizvedb ter poizvedbe primerljive s poizvedbami, ki smo jih izvajali v podjetju.

  Enostavna poizvedba:

  SELECT * FROM STRANKA WHERE STRANKA_ID=5

  Pri testiranju sem v primeru uporabe orodja Hibernate in DataNucleus uporabljal metode za pridobitev objekta glede na primarni ključ, ki so nam na voljo znotraj

JDO in JPA API-jev, pri orodju MyBatis pa, ker nam orodje omogoča le takšen način, sem napisal celotno poizvedbo.

Zahtevnejša poizvedba je bila pri vseh orodjih napisana v SQL jeziku:

```
SELECT c.tel_st, COUNT(o.dat_nar) FROM stranka c, nar_info o, nar_lin ol
WHERE c.title like '%Mr%' AND ol.nar_lin_id=o.nar_lin_id2 AND o.dat_nar
> '2004-03-03'
GROUP BY  c.tel_st
```

Pri primerjavi sem uporabil javanski urejevalnik Eclipse ter podatkovno bazo Postgres, v kateri sem imel 6 medsebojno odvisnih tabel.

V tabeli stranka, kjer sem izvedel enostavno poizvedbo, je bilo notri 15 zapisov, v tabelah, ki so bile udeležene pri naprednejši poizvedbi pa še 12, 10 in 8 zapisov.

3.2 Rezultat primerjave

Pri primerjanju po zgoraj navedenih kriterijih, sem pri vseh treh preslikovalnih orodjih uporabljal iste modelne razrede. Prišel sem do naslednjih ugotovitev:

3.2.1 Kriterij »Število vrstic kode in datotek ter čas pri vzpostavitvi povezave s podatkovno bazo«

Pri MyBatisu so bile za povezovanje s podatkovno bazo in izvedbo enostavne poizvedbe poleg modelnega razreda potrebne še 3 datoteke – nastavitvena xml datoteka, preslikovalna xml datoteka in pa vmesnik. Skupaj je to pomenila 45 vrstic kode. V sami izvedbi klica v glavni funkciji sem potreboval še 7 vrstic kode. Hitrost delovanja kode v glavni funkciji je trajala 408 milisekund.

Hibernate je za isto poizvedbo potreboval manj vrstic kode in tudi manj datotek – poleg modelnega razreda sem potreboval zgolj 1datoteko – konfiguracijsko XML datoteko , ki je zavzela 8 vrstic kode. Hitrost povezovanja na podatkovno bazo in izvedba poizvedbe je trajala 1125 milisekund.

DataNucleus je potreboval za isto poizvedbo 2 datoteki – nastavitveno XML datoteko, ki je zavzela 4 vrstice in pa datoteko z metapodatki, kjer je bilo potrebno opisat razred, ki se bo preslikoval – 8 vrstic. Hitrost povezovanja na podatkovno bazo je trajala 1350 milisekund.

3.2.2 Kriterij »Enostavnost uporabe«

MyBatis je dokaj enostaven za uporabo – ko ugotovimo, za kaj služijo določene datoteke in kakšne so zakonitosti med njimi, je uporaba dokaj enostavna. Potrebno je le paziti na določene sintaktične pravilnosti in pa pravila pri poimenovanju, pri poizvedbah pa imamo opravka z običajno SQL kodo, ki pa je večini programerjev jasna.

Hibernate je tudi precej enostaven za uporabo, toda ker gre za bolj napredno orodje, traja faza učenja nekoliko dlje. Potrebno se je naučiti sintakso Hibernatovega poizvedbenega jezika, uporabljati anotacije in podobno, kar nam v končni fazi olajša povezovanje s podatkovno bazo. Nastavitev je veliko, saj je orodje dokaj kompleksno, toda ko ugotovimo večino zakonitosti Hibernate-a, nam prihrani veliko dela s podatkovno bazo. Hibernate ima tudi to prednost, da je zelo priljubljeno preslikovalno orodje in obstaja veliko pomoči na spletu, če se nam pri delu s Hibernatom zatakne.

DataNucleus je nekoliko manj prijazen za uporabo kot ostali dve preslikovalni orodji. Uporablja JDO API-je, ki so zelo podobni JPA apijem, ki jih uporablja Hibernate. Slabost orodja DataNucleus je slabša pomoč in podpora na spletu. Učna doba za DataNucleus je najdaljša med vsemi tremi, toda ko nam uspe spoznati nekaj zakonitosti tega orodja, postane tudi to izredno enostavno za uporabo.

3.2.3 Kriterij »Hitrost delovanja enostavnih poizvedb ter poizvedbe primerljive s poizvedbami, kot smo jih izvajali v podjetju«

MyBatis je za enostavno poizvedbo na podatkovni bazi, vrnil sem eno vrstico iz določene tabele, potreboval 78 milisekund, za zahtevnejšo poizvedbo pa povprečno 85 milisekund.

Hibernate je za enostavno poizvedbo na podatkovni bazi z uporabo JPA API-jev potreboval 32 milisekund, za zahtevnejšo poizvedbo pa povprečno 34 milisekund.

DataNucleus je za enostavno poizvedbo na podatkovni bazi z uporabo JDO API-jev potreboval 310 milisekund, za zahtevnejšo poizvedbo pa povprečno 370 milisekund.

	Število vrstic kode in datotek, da sem vzpostavil povezavo do podatkovne baze ter povezovalni čas do podatkovne baze	**Enostavnost uporabe**	**Hitrost delovanja enostavnih poizvedb ter poizvedbe primerljive s poizvedbami, kot smo jih izvajali v podjetju**
MyBatis	3 datoteke, 408ms	Precej enostaven, dobro podprt na spletu, malo primerov.	75ms, 85ms
Hibernate	1 datoteka, 1125ms	Izmed primerjanih orodij najbolj enostavno orodje, najbolj podprt na spletu.	32ms, 34ms
DataNucleus	2 datoteki, 1350ms	Srednje enostaven, dobro podprt na spletu, toda malo primerov.	310ms, 370ms

Tabela 1: rezultati pri testiranju.

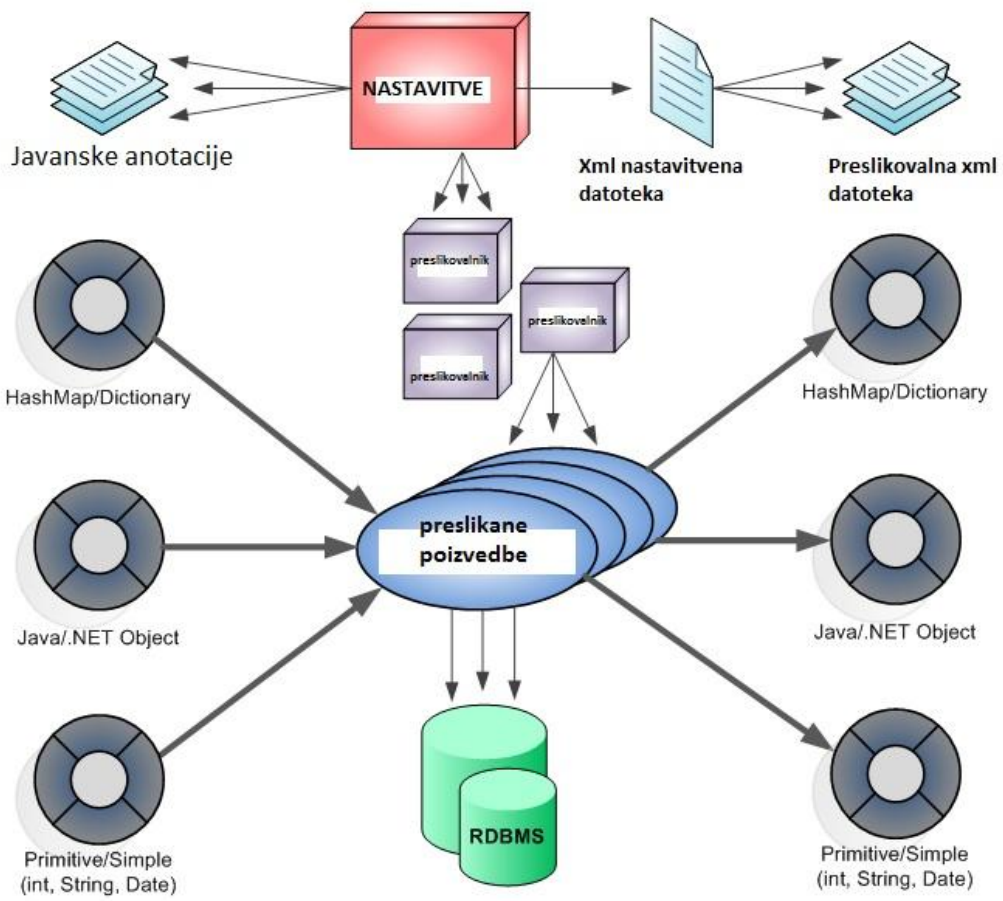
4 Objektno–relacijsko preslikovalna orodja

4.1 MyBatis

MyBatis je razredno programski skelet, ki zagotavlja trajnost podatkom, s podporo za osnovne SQL stavke, procedure in napredna preslikovanja. MyBatis uporablja enostavne XML nastavitvene datoteke in preslika primitivne tipe, preslikovalne vmesnike in Java POJO (osnovne javanske objekte) za zapise iz podatkovne baze.

Središče vsake uporabe MyBatisa je instanca objekta tipa SqlSessionFactory, ki jo pridobimo z uporabo objekta tipa SqlSessionFactoryBuilder, le-ta pa lahko zgradi SQLSessionFactory iz XML nastavitvene datoteke . **[2]**

4.1.1 Delovanje



Slika 2: Slikovna ponazoritev delovanja orodja MyBatis.

4.1.1.1 XML nastavitvena datoteka

XML nastavitvena datoteka vsebuje nastavitve za osnovo delovanja MyBatis sistema ter tudi podatkovni vir za pridobitev povezave z bazo in transactionManagerja za nastavitev, kako naj se obravnavajo transakcije.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
    PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-config.dtd">
<configuration>
  <environments default="development">
      <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="${driver}"/>
        <property name="url" value="${url}"/>
        <property name="username" value="${username}"/>
        <property name="password" value="${password}"/>
      </dataSource>
    </environment>
  </environments>
  <mappers>
    <mapper resource="org/mybatis/example/BlogMapper.xml"/>
  </mappers>
</configuration>
```

Slika 3: primer XML nastavitvene datoteke.

MyBatis nastavitvene XML datoteke vsebujejo nastavitve, ki določajo lastnoti delovanja MyBatis orodja. Struktura posamezne datoteke je sledeča:

- **Lastnosti** – lastnosti, kot sta uporabniško ime in geslo.
- **Nastavitve** – nastavimo lahko, če se uporablja predpomnilnik, omogočimo leno nalaganje...

- **Aliasi tipov** – nastavimo lahko aliase – krajša imenovanja za javanske razrede, ki jih uporabljamo zgolj v preslikovalnih datotekah.
- **Urejevalniki tipov** – z njimi nastavimo, kako se preslikajo tipi iz podatkovne baze v javansko aplikacijo.
- **Razred tipa ObjectFactory**
- **Vtiči**
- **Tip okolja** – v primeru, ko imamo več okolij z različnimi podatkovnimi bazami, lahko nastavimo tip okolja.
- **Preslikovalnik**
- **Upravljalec transakcij** – MyBatis pozna dve vrsti upravljavca transakcij – JDBC in pa MANAGED.
- **JDBC nastavitev** - potrditev ali pa razveljavitev sprememb na podatkovni bazi se izvede direktno. Odvisna je od povezave s podatkovnim virom. Če je vrednost nastavitve nastavljena na MANAGED odločimo o tem, kdaj se izvede potrditev in pa razveljavitev v programski, javanski kodi, orodje MyBatis nam le odpira ter zapira povezavo s podatkovno bazo.

Podatkovni vir

Z vrednostjo elementa podatkovni vir (»DataSource«) nastavimo podatkovni vir, na katerega se bomo prek JDBC povezave povezovali. Poznamo tri vrednosti elementa podatkovni vir:

- **UNPOOLED** – Povezava s podatkovnim virom se odpre vsakič, ko se želimo povezati do podatkovne baze. Različne podatkovne baze so pri taki nastavitvi tudi performančno različne.

  Temu tipu povezave s podatkovnim virom lahko določimo štiri atribute: jdbc gonilnik, spletni naslov podatkovnega strežnika, geslo in defaultTransactionIsolationLevel.

- **POOLED** – Ta implementacija podatkovnega vira ustvari bazen JDBC povezavnih objektov, da se izogne daljšemu času povezovanja in avtentikacije, ki je potreben za izdelavo nove instance tipa Connection. To je priljubljen pristop za hitro odzivne

spletne aplikacije, da se doseže hitrejše odzive. Pri pooled povezavi podatkovnega vira lahko nastavimo več atributov – od tega, koliko živih povezav lahko obstaja istočasno, koliko neaktivnih povezav lahko sploh obstaja, če se pošlje posamezni povezavi ping signal...

- **JNDI** – ta tip nastavitve podatkovnega vira se uporablja, ko imamo v sami aplikaciji na voljo uporabo z zbiralnika kakor je Spring, ali pa aplikacijski strežnik, na katerega lahko nastavimo podatkovni vir centralno ali pa zunanje in dodamo referenco. Na njih se poda referenco kot JNDI lastnost.

4.1.1.2 Preslikovalna XML datoteka

Nastavitve za objekt tipa SQLConnectionFactory pa lahko nastavimo tudi znotraj same javanske kode .

Ko imamo nastavljen objekt SqlSessionFactory, lahko iz njega dobimo instanco objekta SqlSession. SqlSession vsebuje vse metode, ki jih potrebujemo za izvedbo SQL poizvedb na podatkovni bazi.

Ukazi, katere želimo izvesti na podatkovni bazi, so napisani v XML datoteki (povezovalnik).

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper
    PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
    "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
<mapper namespace="org.mybatis.example.BlogMapper">
  <select id="selectBlog" parameterType="int" resultType="Blog">
    select * from Blog where id = #{id}
  </select>
</mapper>
```

Slika 4: Primer povezovalne XML datoteke.

V eno preslikovalno XML doteko lahko napišemo več SQL poizvedb. Posamezna poizvedba ima svoje ime (v zgornjem primeru »selectBlog«) v imenskem prostoru »org.example.BlogMapper«.

Delovanje orodja MyBatis temelji na povezovalnih stavkih, ki so zapisani v povezovalnih XML datotekah. Uporaba takih stavkov napram kodi, pisani v javanski aplikaciji sami, z uporabo JDBC povezovalnih objektov, povzroči zmanjšanje količine vrstic kode v javanskem delu aplikacije, ki je potrebna za povezovanje do podatkovne baze za približno 95 procentov. Povezovalne datoteke vsebujejo elemente si po shemi morajo sledit v tem vrstnem red:

- **predpomnilnik** – nastavitve v primeru, da želimo posamezni imenski prostor naložiti v predpomnilnik. Prek predpomnilnika Mybatis ve, kdaj zakleniti vrstico v bazi, kdaj ne, kdaj jo lahko sprosti in podobno. Vse to je izjemno pomembno pri razvoju aplikacij, kjer več uporabnikov uporablja iste podatke.
- **referenca predpomnilnika** – referenca nastavitev za nalaganje v predpomnilnik iz drugega imenskega prostora.
- **resultMap** – najbolj zapleten in najpomembnejši elelemt, ki opisuje, kako naložiti objekte iz rezultatov pri poizvedbi na podatkovni bazi.
- **sql** – del SQL stavka, ki ga lahko prilepimo v katerokoli SQL poizvedbo kot procedure.
- **insert** – povezovalni INSERT stavek na podatkovni bazi.
- **update** – povezovalni UPDATE stavek na podatkovni bazi.
- **delete** – povezovalni DELETE stavek na podatkovni bazi.
- **SELECT** – povezovalni SELECT stavek na podatkovni bazi.

```
<select id="selectPerson" parameterType="int" resultType="hashmap">
  SELECT * FROM PERSON WHERE ID = #{id}
</select>
```

Slika 5: primer SELECT stavka v preslikovalni XML datoteki.

Vsak izmed elementov, ki se uporabljajo za preslikanje z bazo vsebuje parametre »id«, ki nam pove, kako se kliče to funkcijo pri javanskem delu razvoja, »parametertype«, ta pa nam pove, kakšen bo tip vhodne spremenljivke v stavek ter »resultType«, ki označuje, kakšen bo javanski tip izhodne spremenljivke.

»Result-map« uporabimo, ko tip izhodne spremenljivke ni osnoven. V javanskem delu aplikacije izgleda resultMap kot običajen javanski razred (POJO). Ti atributi se preslikajo na dobljene stolpce iz stavka Select, na primer polje ID iz tabele v podatkovni bazi se preslika v atribut ID v resultMapu.

```
package com.someapp.model;
public class User {
  private int id;
  private String username;
  private String hashedPassword;

  public int getId() {
      return id;
  }
  public void setId(int id) {
      this.id = id;
  }
  public String getUsername() {
      return username;
  }
  public void setUsername(String username) {
      this.username = username;
  }
  public String getHashedPassword() {
      return hashedPassword;
  }
  public void setHashedPassword(String hashedPassword) {
      this.hashedPassword = hashedPassword;
  }
}
```

Slika 6: Primer Javanskega zrna, v kateri se preslikajo rezultati poizvedbe na podatkovni bazi.

```
<select id="selectUsers" parameterType="int"
        resultType="com.someapp.model.User">
        select id, username, hashedPassword
        from some_table
        where id = #{id}
</sql>
```

Slika 7: Primer MyBatisovega Select stavka, ki uporabi zgornji ResultMap kot ResultSet

Rezultat pri select stavku mora dobiti poimensko natančno iste parametre kot so v javanskem modelnem razredu, drugače se poizvedba na relacijski podatkovni bazi ne bo preslikala v objekt. V primeru, da dobljeni stolpci, dobljeni kot rezultat poizvedbe nimajo istih imen, uporabimo ključno besedo »AS« in podamo dobljenemu stolpcu podamo ime, ki je potrebno. V primeru, da modelni razred vsebuje atribut tipa seznam, lahko pripadajoči SELECT stavek prilagodimo tudi temu, saj obstajajo elementi znotraj SELECT stavkov, ki nam omogočijo:

- če uporabimo element Constructor, lahko prek ResultSeta direktno instanciramo objekt, MyBatis mora le prek atributov zaznati za kater objekt gre.
- element association nam omogoči povezovanje z že napisanimi SELECT stavki. Z drugimi SELECT stavki se povezuje prek elementa ID, ki se mora ujemati s stolpcem ID v drugem SELECT stavku.
- element Collection se uporabi, ko imamo v javanskem zrnu kot atribut podan drug objekt. Znotraj Collection elementa se uporabi drug SELECT stavek, ki vrne objekt nekega tipa, ki je podan tudi kot atribut v javanskem zrnu, ki ga želimo napolniti. Deluje podobno kot Association, le da vrne skupino rezultatov in da je potrebno še podati tip, ki ga gnezdeni SELECT vrne s parametrom »ofType«.
- discriminator – omogoča vračanje različnih tipov, ker preverja rezultate iz gnezdenih SELECT stavkov. Je MyBytisov ekvivalent javanskemu preklopnem stavku.

```
<!-- Very Complex Result Map -->
<resultMap id="detailedBlogResultMap" type="Blog">
  <constructor>
    <idArg column="blog_id" javaType="int"/>
  </constructor>
  <result property="title" column="blog_title"/>
  <association property="author" column="blog_author_id" javaType=" Author">
    <id property="id" column="author_id"/>
    <result property="username" column="author_username"/>
    <result property="password" column="author_password"/>
    <result property="email" column="author_email"/>
    <result property="bio" column="author_bio"/>
    <result property="favouriteSection" column="author_favourite_section"/>
  </association>
  <collection property="posts" ofType="Post">
    <id property="id" column="post_id"/>
    <result property="subject" column="post_subject"/>
    <association property="author" column="post_author_id" javaType="Author"/>
    <collection property="comments" column="post_id" ofType=" Comment">
      <id property="id" column="comment_id"/>
    </collection>
    <collection property="tags" column="post_id" ofType=" Tag" >
      <id property="id" column="tag_id"/>
    </collection>

     <discriminator javaType="int" column="draft">
       <case value="1" resultType="DraftPost"/>
     </discriminator>
   </collection>
 </resultMap>
```

Slika 8: primer zapletenejšega ResultSeta.

4.1.1.3 Dinamični SQL

Ena od funkcionalnosti MyBatisa je dinamično polnjenje SQL poizvedb. Znotraj MyBatis SELECT stavka lahko uporabimo pogoje in zanke, nize lahko celo krčimo.

```
<select id="selectPostIn" resultType="domain.blog.Post">
    SELECT *
    FROM POST P
    WHERE ID in
    <foreach item="item" index="index" collection="list"
        open="(" separator="," close=")">
          #{item}
    </foreach>
  </select>
```

Slika 9: primer foreach stavka znotraj MyBatis select-a

4.1.2 Prednosti in slabosti

Orodje MyBatis nam s svojimi funkcionalnostmi omogoča krajšanje razvojnega časa ne glede na obsežnost aplikacije, ki jo razvijamo. Od ostalih dveh primerjanih preslikovalnikov se razlikuje v tem, da se ne drži nobenega standardnega API-ja za preslikovanje in ima svoj okvir delovanja – zaradi te lastnosti ga nekateri ne štejejo med objektno-preslikovalna orodja.

Uporaba MyBatisa nam omogoča trajnost modelnih razredov. Atribute v modelnih razredih napišemo glede na dobljene stolpce pri poizvedbi. Jezik za poizvedbe na podatkovni bazi je običajen SQL jezik – s sintakso, kot je za izbrani tip podatkovne baze potrebna.

Razčlenitev po funkcionalnosti na XML datoteke nam omogoča večji nadzor nad deli aplikacije, kjer se povezujemo s podatkovno bazo.

Orodje MyBatis trenutno podpira okolje Java in pa .Net.

4.2 Hibernate

Hibernate je objektno-relacijsko preslikovalno orodje, ki se uporablja se v podatkovni plasti in omogoča trajnost podatkom v atributih preslikanih, javanskih modelnih razredov, iz tabel v podatkovno bazo. Drži se standardov JPA, torej lahko Hibernate uporabljamo le za preslikave iz relacijskih podatkovni baz in obratno. Prek anotacij v modelnih razredih določimo, kako se modelni razredi preslikajo v tabele v podatkovni bazi, kateri je njihov ID atribut, v kakšnih razmerjih so tabele med seboj (npr. ena-več...). Uporaba poizvedbenega jezika orodja Hibernate – HQL omogoča neodvisnost SQL poizvedb glede na sintaktične zahteve različnih vrst podatkovnih baz. **[14]**

4.2.1 Delovanje

4.2.1.1 Hibernate konfiguracijska datoteka

Standardno ime za Hibernate konfiguracijsko datoteko je hibernate.cfg.xml. V konfiguracijsko datoteko se vpiše nastavitvene vrednosti, da dosežemo željen način delovanja orodja Hibernate. **[3]** Nastavitveni elementi v XML konfigaracijski datoteki so:

Database conn setings – potrebno je podati podrobne podatke za te nastavitve – kje se nahaja podatkovna baza, uporabniško ime, geslo in kje se nahaja gonilnik za JDBC povezavo do naše podatkovne baze.

Database connection pool size – število hkratnih povezav na podatkovno bazo

Dialekt –nastavitev, ki orodju Hibernate pove, kakšen jezik potrebuje, da komunicira s podatkovno bazo – zaradi sintaktičnih razlik med ukazi med bazami (npr. razlike med Oracle notacijo in MySql notacijo). V knjižnici Hibernate imamo omogočene dialekte za večino tipov relacijskih podatkovnih baz.

Second level cache – uporaba drugonivojskega medpomnilnika – nastavitev, ki preprečuje hkratno zajemanje istih podatkov na podatkovni bazi.

showSQL – nastavitev, ki določi ali naj se izpisuje v konzoli stavke, ki jih je Hibernate izvedel ali ne.

Hb2mdll.auto – nastavitev, ki nam omogoča, da ob generiranju objektov, ki še nimajo korespondenčnih tabel v podatkovni bazi, orodje zgenerira korespondenčne tabele. Če tabele že obstajajo, jih lahko zbrišemo in na novo skreiramo, lahko pa podatke samo dodajamo že obstoječim tabelam.

Mapping class – modelni razredi, v katere se bo rezultate preslikovalo – vsi modelni razredi, ki jih orodje Hibernate uporablja, morajo biti omenjeni znotraj tega nastavitvenega elementa.

4.2.1.2 Modelni razred

V vsakem razredu so atributi, za katere moramo zgenerirati get in set metode. Modelni razredi morajo biti zabeleženi v preslikovalni class nastavitvi v konfiguracijski datoteki. Znotraj orodja Hibernate-a lahko uporabljamo anotacije.

```
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class UserDetails {
    @Id
    private int userId;
    private String userName;
    public int getUserId() {
        return userId;
    }
    public void setUserId(int userId) {
        this.userId = userId;
    }
    public String getUserName() {
        return userName;
    }
    public void setUserName(String userName) {
        this.userName = userName;
```

Slika 10: Primer modelnega objekta.

4.2.1.3 Možnosti

Hibernate zna preslikovati v obe smeri – iz objektov v tabele in tabele v objekte.

Da iz enostavne tabele napolnimo objekt, nam ni potrebno postoriti veliko stvari – potrebujemo objekt tipa SessionFactory – del Hibernate API-ja, ki nam vzpostavi povezavo.

Objekt tipa SessionFactory obstaja le en na aplikacijo. Objekt vzpostavlja seje. Če želimo kaj shraniti, se prek seje povežemo s podatkovno bazo. Session factory-ju tudi podamo pot do konfiguracijske datoteke. Druga stvar, ki jo potrebujemo, je objekt tipa Session – objekt, prek katerega shranjujemo objekte v podatkovno bazo in dobivamo objekte iz podatkovne baze. **[7]**

```
public class HibernateTest {

    /**
     * @param args
     */
    public static void main(String[] args) {
        UserDetails user = new UserDetails();
        user.setUserId(1);
        user.setUserName("First User");

        SessionFactory sessionFactory = new Configuration().configure().buildSessionF
        Session session = sessionFactory.openSession();
        session.beginTransaction();
        session.save(user);
        session.getTransaction().commit();

    }

}
```

Slika 11: osnovni postopek za povezovanje z podatkovno baze

Shranjujemo lahko tudi zbirke. Če prek anotacij podamo zahtevane vrednosti, nam Hibernate tudi avtomatsko zgradi tabele in povezave med njimi. Prav tako zna sam zgenerirati povezave ena – več (prek vmesne tabele) in ena – ena. Simulirati zna tudi dedovanje na podatkovni baze.

4.2.1.4 Crud operacije

Hibernate ima funkcionalno pokrite vse štiri CRUD (create, retrieve, update, delete) operacije:

Create (shranjevanje): Za vpisovanje novih podatkov v podatkovno tabelo v podatkovni bazi. Pri uporabi orodja Hibernate uporabimo ukaz Session.save. Shranjujemo izbrani modelni tip.

```
SessionFactory sessionFactory = new Configuration().configure().buildSessionFac
Session session = sessionFactory.openSession();
session.beginTransaction();

for(int i=0; i<10; i++) {
    UserDetails user = new UserDetails();
    user.setUserName("User " + i);
    session.save(user);

}

session.getTransaction().commit();
session.close();
```

Slika 12: primer shranjevanja podatkov v podatkovno bazo.

Retrieve (pridobivanje podatkov): Za pridobivanje podatkov iz podatkovne baze uporabimo ukaz »get«. Kot parametra moramo vnesti, kakšen tip razreda shranjujemo – pravi modelni razred in pa vrednost primarnega ključa tabele, iz katere pridobivamo podatke.

```
SessionFactory sessionFactory = new Configuration().configure().buildSessionFa
Session session = sessionFactory.openSession();
session.beginTransaction();

UserDetails user = (UserDetails) session.get(UserDetails.class, 6);


session.getTransaction().commit();
session.close();
```

Slika 13: pridobivanje podatkov iz podatkovne baze.

Update (osveževanje podatkov): Za klic osveževanja podatkov uporabimo ukaz Session.update(). Procedura deluje kot navadna set metoda, le da kasneje klic objekta Session izvede klic metode UPDATE na podatkovni bazi.

```
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();
session.beginTransaction();

UserDetails user = (UserDetails) session.get(UserDetails.class, 5);
user.setUserName("Updated User");
session.update(user);


session.getTransaction().commit();
session.close();
```

Slika 14: klic osveževalne metode.

Delete (brisanje podatkov): Za brisanje podatkov iz podatkovne baze uporabimo metodo »delete«. Tej metodi podamo objekt, ki se ga želimo znebiti v podatkovni tabeli, Hibernate-u ni potrebno podati vrednosti primarnega ključa – glede na podani objekt ga orodje prepozna in uporabi samo.

```
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();
session.beginTransaction();

UserDetails user = (UserDetails) session.get(UserDetails.class, 6);
session.delete(user);


session.getTransaction().commit();
session.close();
```

Slika 15: klic funkcije delete.

4.2.2 HQL

Za preslikovanje zahtevnejših poizvedb v Hibernate-u uporabimo Hibernate-ov jezik HQL – hibernate query language. Za uporabo HQL-a potrebujemo inicializacijo osnovnih Hibernate-ovih razredov – session factory s konfiguracijami in pa objekt Session. Znotraj Session objekta pokličemo metodo »create query«. Pri pisanju poizvedb ne kličemo neposredno tabel v podatkovni baze, ampak objekt, ki je preslikan na tabelo glede na naše želene podatke. Prav tako pri poizvedbi za določen stolpec iz podatkovne baze ne uporabimo imena stolpca v podatkovni bazi, temveč klic get funkcije za želeni atribut v korespondenčnem modelnem razredu. Hibernate nam vedno vrne rezultat naše ročno napisane poizvedbe kot seznam. Nastavimo lahko indeks prvega rezultata v seznamu in maksimalno število zadetkov. Seveda lahko poizvedbi HQL dinamično podamo tudi iskalne parametre tudi prek substitucije parametrov – Query objektu podamo vrednost in pa tip parametra.

```
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();
session.beginTransaction();
String minUserId = "5";
String userName = "User 10";
Query query = session.createQuery("from UserDetails where userId > ? and userName = ?");
query.setInteger(0, Integer.parseInt(minUserId));

List<UserDetails> users = (List<UserDetails>) query.list();
session.getTransaction().commit();
session.close();
```

Slika 16: primer uporabe HQL

Parametre lahko podajamo tudi s predpono »:« in pa imenom spremenljivke, katere vrednost želimo uporabiti v poizvedbi.

V primeru, da želimo imeti ločeno javansko kodo in pa poizvedbe na podatkovno bazo, lahko poizvedbe pišemo tudi v svoji datoteki. Za uporabo te funkcije moramo podati pred poizvedbo javansko anotacijo tipa »namedquery«. Znotraj poizvedbe lahko uporabljamo substitucijo tako da, ko želimo dinamično napolniti vrednost, dodamo znak »?«. Ob zagonu aplikacije se znaki »?« zamenjajo z vrednostmi.

```
@Entity
@NamedQuery(name="UserDetails.byId", query="from UserDetails where userId = ?")
```

Slika 17: primer anotacije, ki vsebuje HQL poizvedbo.

```
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();
session.beginTransaction();

Query query = session.getNamedQuery("UserDetails.byId");
query.setInteger(0, 2);

List<UserDetails> users = (List<UserDetails>) query.list();
session.getTransaction().commit();
session.close();
```

Slika 18: primer klica prek anotacije klicane HQL poizvedbe.

Vse omenjene mehanizme za pridobivanje podatkov lahko napišemo tudi v mapirni datoteki – znotraj XML bloka »hibernate mapping«, kjer lahko kličemo tudi SQL procedure.

Preko teh dveh načinov pridobivanja podatkov iz podatkovne baze smo omejeni – ne moremo izvesti kompleksnejših poizvedb, pri le-teh pa koda postaja vedno manj obvladljiva. Zato se uporablja Criteria API. Namesto Query objekta uporabimo Criteria objekt. Uporabimo metodo Session.createCriteria. Objekt tipa Criteria še vedno ne uporablja direktno podatkovne

baze, pač pa še vedno uporabi Hibernatov API. Criteria je še najbolj podobna Where pogoju. Znotraj objekta tipa Criteria lahko specificiramo pogoje in omejitve. Pravzaprav pišemo kriterij pri pridobivanju podatkov iz podatkovne baze na navzven osnovni način orodja Hibernate.

```
SessionFactory sessionFactory = new Configuration().configure().buildSessionFactory();
Session session = sessionFactory.openSession();
session.beginTransaction();

Criteria criteria = session.createCriteria(UserDetails.class);
criteria.add(Restrictions.eq("userName", "User 10"));


List<UserDetails> users = (List<UserDetails>) criteria.list();
session.getTransaction().commit();
session.close();
```

Slika 19: primer uporabe Criteria objekta.

Prek omejitev znotraj objekta Criteria lahko izvedemo praktično vsako SQL poizvedbo (vmes, največja vrednost, število vrnjenih vrstic ipd.). Lahko podamo tudi primer, kateri objekt želimo preslikati iz podatkovne baze in ga shraniti v javanski objekt prileženga tipa.

4.2.2.1 Shranjevanje v predpomnilniku:

Hibernate omogoča več nivojev shranjevanja v predpomnilniku:
Prvi nivo: nivo seje – seja se ohranja za vso kodo, ki se izvede znotraj ene seje.

Drugi nivo: Hibernate nam omogoča prehode med odprtimi sejami, ki so že shranjene v predpomnilniku, združevanje sej, povežemo pa lahko tudi seje v aplikacijah, ki upravljajo z istimi podatki.

Prek shranjenih objektov v sejah Hibernate zazna, če že ima nek določen objekt v seji, zato ne dovoli drugi seji, kjer želimo uporabiti iste podatke in jih spremeniti, da bi izvedla posodobitveno funkcijo nad istimi podatki. Hibernate zazna, da je isti objekt že v predpomnilniku, drugi seji pa onemogoči dostop do njega. Na kratko: dokler se za objekt uporablja drugonivojski tip hranjenja v predpomnilniku so podatki zaklenjeni za obdelavo in se, dokler se seja, ki jih je prva želela uporabiti ne zapre, osvežujejo in pa zaklepajo.

Drugonivojsko medpomnenje lahko nastavimo tudi drugače – znotraj konfiguracijske datoteke in pa kot anotacijo v modelnih razredih z nastavitvijo na vrednost »cachable«. Ko so podatki že v medpomnilniku, jih zaklene. Ta možnost nam pride prav, ko imamo več delujočih instanc aplikacije, kjer vsi uporabniki uporabljajo iste podatke. Za lažjo predstavo si vzemimo primer blagajne v kinu – enega sedeža ne moremo ponuditi večim kupcem vstopnice.

4.2.3 Prednosti in slabosti

Objektno–relacijsko orodje Hibernate se od orodja MyBatis precej razlikuje. Preslikovanje objektov v tabele je izvedeno skladno s standardom JPA. Zaradi uporabe standardov JPA nudi večje število uporabnih funkcij.

Hibernatova preslikovanja delujejo na podlagi refleksije – ob prevajanju se modelnim razredom dodelijo funkcije, ki zagotavljajo trajnost atributom in omogočajo preslikovanje.

Orodje omogoča preverjanje trajnosti podatkov in njihovih vrednosti v medpomnilniku na okolju, kjer se izvršuje javanska koda – ta funkcionalnost nam pride prav, ko želimo zakleniti posamezno vrstico podatkov iz tabele v podatkovni bazi – ni potrebno ročno, prek poizvedb zaklepati podatkov (npr. ukaz SELECT FOR UPDATE – ukaz, ki podatkovno bazo prisili k zaklepanju podatkov).

Prek get metode nam orodje omogoča dostop do vrednosti iz želene tabele glede na vrednost njenega primarnega ključa. Pri pridobivanju podatkov iz podatkovne baze, pri nekoliko zahtevnejših poizvedbah, lahko uporabimo tudi poizvedbeni jezik orodja Hibernate – HQL ali pa napišemo običajno SQL poizvedbo.

Pri razvoju aplikacij s povečanim hkratnim dostopom do iste podatkovne baze iz strani uporabnikov si s Hibernatom lahko pomagamo, toda zaradi količine prostora na pomnilniku, ki je uporabljen pri prvem nalaganju, lahko uporaba Hibernate-a privede do pomanjkanja pomnilniškega prostora. Pri aplikacijah, kjer zasedenost pomnilniškega prostora ne pomeni resnejše ovire, pa se po mojih izkušnjah Hibernate izkaže za zelo koristen pripomoček pri razvoju ter nam opazno skrajša čas razvoja.

4.3 DataNucleus

DataNucleus je objektno–relacijsko preslikovalno orodje, ki se pri svojem delovanju drži standardov JDO, imamo pa celo možnost izbirati med JDO in JPA.

4.3.1 Delovanje

Za samo uporabo JDO iz uporabnikovega vidika, JDO vsebuje vmesnike:

- **Persistance manager** – komponenta, ki omogoča trajnost objektov, izvedbo poizvedb in transacij.
- **Query** – Komponenta, odgovorna za poizvajanje na podatkovni bazi in vračanje instanc objektov, napolnjenih s podatki.
- **Transaction** – Komponenta za odpiranje in dokončanje transakcij.

JDO se razvija kot Java Specification Request v Java Community procesu. Prvi JDO je bil JDO 1.0, trenutni pa je JDO 3.0.

Trenutno za razvoj projekta skrbi podjetje Apache, orodje je odprtokodno in brezplačno.

4.3.2 Podprti podatkovni zbiri

DataNucleus podpira uporabo obeh glavnih APIjev za trajnost – JDO in JPA. Omogoča nam celo, da med razvojem preklapljamo med obema APIjema. Orodje nam omogoča izvajanje poizvedb, ne nujno na relacijski podatkovni bazi, pač pa tudi na drugih zbirališčih podatkov - db4o, NeoDatis, Excel, OOXML, ODF, XML, JSON, Amazon S3, GoogleStorage, HBase, MongoDB in LDAP.

Če v aplikaciji uporabimo relacijsko podatkovno bazo, lahko prek JDO uporabimo persistenčne objekte z uporabo JDBC. Omogoča nam, da ne pišemo skozi razvoj aplikacije pri povezavi s podatkovno bazo vedno iste povezovalne ukaze in se s trajnostjo ukvarjamo ročno.

Pri uporabi Datanucleus-a lahko uporabimo tudi JPA – standardizirani persistenčno API in del EJB3 specifikacij. To nam tudi omogoča razvoj navadnih javanskih objektov, atributom teh objektov pa zagotavlja trajnost z uporabo standardnih APIjev.

Uporaba DataNucleusa nam omogoča preslikovanje objektov v tabele iz podatkovne baze in obratno.

4.3.3 Tipi JDO objektov

DataNucleus, prek standarda JDO zagotavlja transparentno trajnost objektov. Znotraj JDO obstajajo trije tipi razredov:

- **trajnostno zmožni** (persistence capable) – vrsta razredov, ki jim zagotovimo trajnost. JDO zagotavlja mehanizme za zagotavljanje trajnosti in so jedro JDO okolja.
- **trajnostno zavedni** (persistance aware) – tipi razredov, ki upravljajo s trajnostno zmožnimi instancami objektov prek manipulacije z njihovimi atributi. Ti razredi so tipično razširjeni z meta podatki.
- **običajni** – tipi razredov, ki sami po sebi nimajo lastnosti trajnosti ter se jih trajnost ne dotika. To so povsem običajni javanski razredi.

Razredi so definirani kot trajnostno zmožni ali pa trajnostno zavedni prek XML metapodatkov, ali pa od JDO 2.1 naprej, prek javinih anotacij.

4.3.3.1 Podprti javanski tipi spremenljivk

Za preslikavo polj iz tabel v podatkovni bazi mora JDO vedeti, v kakšen javanski tip se posamezni bazni tip preslika. JDO vsebuje seznam, kateri tipi so potrebni za preslikavo polj iz podatkovne baze. Razdelimo jih lahko na dve kategoriji:

Prvorazredni tipi: objekt, katerega lahko referenciramo in ima ID atribut. JDO potrebuje implementacijo takih tipov objektov za uporabo trajnostno zmožnih tipov, kakor tudi v pomoč objektom/vmesnikom – poljem, ki referencirajo trajnostno zmožnim tipom.

Drugorazredni tipi: Atributi nekaterih tipov so lahko primarni ključi v tabelah, drugi ne, nekateri so lahko trajnostni, drugi ne.

Java Type	DFG?	Persistent?	PK?
boolean	✓	✓	✓
byte	✓	✓	✓
char	✓	✓	✓
double	✓	✓	✗
float	✓	✓	✗
int	✓	✓	✓
long	✓	✓	✓
short	✓	✓	✓
boolean[]	✗	✓	✗
byte[]	✗	✓	✗
char[]	✗	✓	✗
double[]	✗	✓	✗
float[]	✗	✓	✗
int[]	✗	✓	✗
long[]	✗	✓	✗
short[]	✗	✓	✗

Slika 20: JDO tipi atributov.

4.3.3.2 Metadata nastavitve

Za podrobnejše nastavljanje, kako naj se persistenčno omogočeni objekti obnašajo, name je omogočeno podrobnejše nastavljanje znotraj XML datotek – kot JDO-jevi meta podatki.

JDO pričakuje katerekoli podatke, ki bodo služili kot nastavitve – poti do datotek, ki jih aplikacija uporablja in podobno. Na primer, če imamo razred MojPrimer, bo JDO želel ta razred najti in ga bo iskal, dokler ga ne najde.V metapodatkih mu pot do razreda podamo. V metapodatkih podamo JDO-ju tudi preslikovalne inormacije – lahko jih shranimo kot datoteko tipa .orm., lahko pa tudi kot datoteko tipa .jdo.

Metapodatke pa lahko od JDO 2.1 naprej shranimo tudi kot javine anotacije.

4.3.4 JDOHelper

JDOHelper je standardni pripomoček, ki nam omogoča dostop do uporabnih delov JDO persistenčega procesa. Vsebuje metodo za začetek persistenčnega procesa – PersistanceManagerFactory ali na kratko PMF in pa operacije, ki jih izvaja na trajnosti ter metode, ki skrbijo za življenjski cikel teh operacij.

4.3.5 Persistance manager Factory

Vsaka aplikacija, ki uporablja JDO bo potrebovala vsaj eno instanco razreda PersistanceManagerFactory. Normalno se uporablja ena instanca na podatkovno bazo, ki jo uporabljamo. Omogoča nam odstop do persistenceManagerjem. Znotraj objekta tega tipa podamo pot do podatkovne baze in ostale stvari, ki jih potrebujemo za dostop do baze. Namesto, da PMFju nastavljamo vse preko javanske kode, lahko uporabimo datoteko tipa properties.

4.3.6 Trajnostni urejevalnik

Preko objekta tipa PersistentManager izvajamo vse »CRUD« operacije, prav tako pa lahko napišemo znotraj njega poizvedbo na podatkovno bazo. Z enim PersistentManagerjem obravnavamo eno samo transakcijo.

4.3.7 Transakcija

Znotraj JDO uporabljamo dva tipa transakcij: transakcije, ki zaklenejo vse podatke v podatkovni bazi in jim omogočijo uporabo šele, ko je transakcije konec – pesimistične transakcije. Druga vrsta transakcij pa je optimistična in podatkov ne zaklepa.

4.3.8 Vračanje objektov

Objekte vračam z metodo get, kateri podamo tip razreda, katerega vračamo. Če želimo dobiti vse objekte iskanega tipa (vse vrstice v podatkovni tabeli), uporabimo funkcijo getExtent – z javansko iteracijo se potem sprehodimo čez vse rezultate. Za malce naprednejše poizvedbe uporabimo tip Criteria – kriteriji, ki se uporabijo pri poizvedbi.

4.3.9 JDOQL

JDOQL je JDO-jev način izvajanja poizvedb na podatkovni bazi s pisanjem JDOQL kode. Sam princip delovanja je podoben delovanju Hibernate-ovega HQL – pišemo poizvedbe prek razredov. V nastavitveni datoteki moramo le nastaviti in ročno preslikati mapirni razred. JDOQL vsebuje vse, kar potrebujemo za zahtevnejše poizvedbe na podatkovni bazi.

```
Declarative JDOQL :
Query q = pm.newQuery(mydomain.Person.class, "lastName == \"Jones\" && age < age_limit");
q.declareParameters("double age_limit");
List results = (List)q.execute(20.0);

Single-String JDOQL :
Query q = pm.newQuery("SELECT FROM mydomain.Person WHERE lastName == \"Jones\"" +
                      " && age < :age_limit PARAMETERS double age_limit");
List results = (List)q.execute(20.0);
```

Slika 21: Primer poizvedbe v JDOQL.

JDOQL kodo podamo običajno v običajnem javinem nizu. Referenciramo lahko tako trajnostne atribute kot tudi netrajnostne atribute. Kodi v JDOQL-u lahko podatke podajamo dinamično prek zunanjih paremetrov.

```
class Product
{
    String name;
    double price;
    java.util.Date endDate;
    ...
}

<jdo>
    <package name="mydomain">
        <class name="Product">
            <field name="name">
                <column length="100" jdbc-type="VARCHAR"/>
            </field>
            <field name="abreviation">
                <column length="20" jdbc-type="VARCHAR"/>
            </field>
            <field name="price"/>
            <field name="endDate"/>
        </class>
    </package>
</jdo>
```

Slika 22: Primer preslikanega razreda "Product" v konfiguracijski datoteki.

Če želimo, pa JDO podpira tudi pisanje kode v običajni SQL poizvedbi.

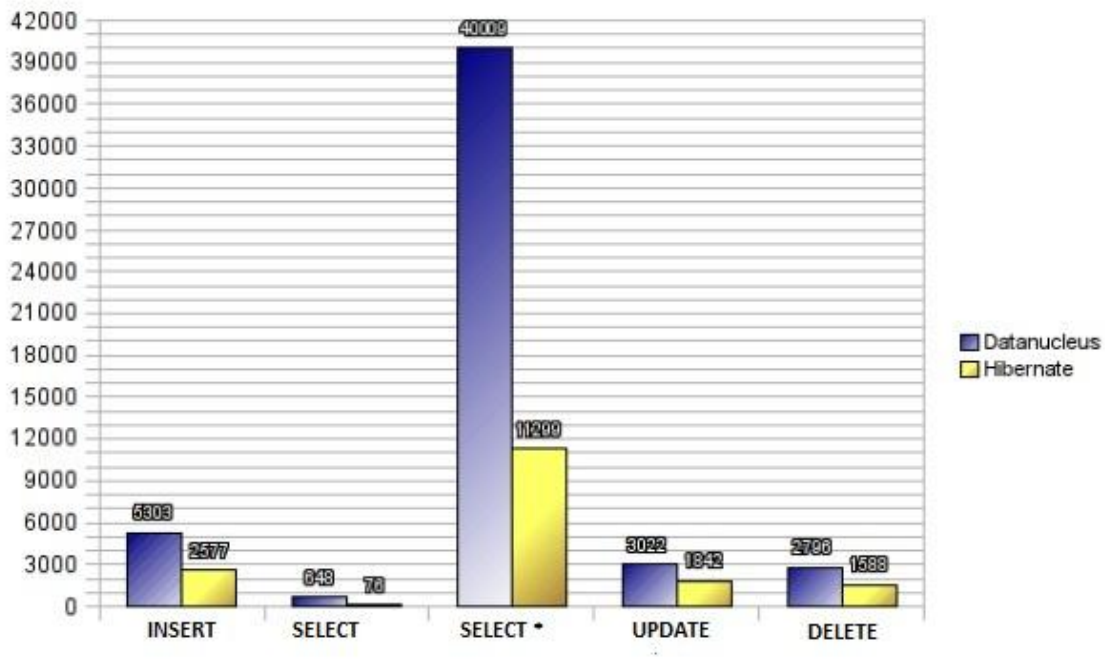
4.3.10 Prednosti in slabosti

DataNucleus je objektno–relacijsko orodje, ki se drži stanarda JDO. Pri samem preslikovanju in zagotavljanju trajnosti razredom in njihovim atributom uporablja razširjanje.

Razširjanje pomeni, da orodje DataNucleus v binarno javansko kodo (objekti .class) pri prevajanju doda bite, ki jih zgenerira glede na vrednost v nastavitveni datoteki

Uporaba orodja DataNucleus je zaradi podobosti JDO in JPA API-jev precej podobna uporabi orodja Hibernate. Orodje omogoča klic get in set metod za pridobivanje rezultatov pri poizvedbah glede na primaren ključ v eni tabeli, pri uporabi kompleksnejših metod pa lahko uporabimo JDOQL – podoben poizvedbeni jezik kot HQL. Pri poizvedbah primerjamo vrednosti atributov razredov, ne pa direktnih vrednosti stolpcev v podatkovni bazi. JDO lahko uporabimo tudi pri pridobivanju podatkov iz zbirnika podatkov, ki ni nujno relacijska podatkovna baza – prek JDO lahko referenciramo podatke iz XML datotek in podobno.

Po nedavnih študijah[17], ko so uporabili tabele, napolnjene z 2047 vrsticami v vsaki tabeli in pri vsaki poizvedbi pognali šestdesetkrat ter iz tega izračunali povprečni čas, je ob primerjavi hitrosti izvajanja poizvedb osnovnih »CRUD« operacij, orodje Hibernate prav v vsakem primeru hitrejše od orodja DataNucleus. Pri pridobivanju rezultatov po identifikacijskem polju v tabeli pa je bil Hibernate celo devetkrat hitrejši.



Slika 23: študija časovne uspešnosti delovanja orodja Hibernate in DataNucleus pri CRUD operacijah.

5 Zaključek

Skozi izdelavo diplomske naloge sem spoznal delovanje objektno–relacijskih preslikovalnikov ter njihovega delovanja. Na podlagi vseh ugotovitev sem prišel do zaključkov:

Hibernate potrebuje za vzpostavitev povezave s podatkovno bazo in nalaganjem potrebnih razredov precej časa (zaradi API-jev povzetih po standardu JPA) – trikrat več kot MyBatis, ampak manj kot DataNucleus. Sama poizvedba na podatkovno bazo poteka približno trikrat hitreje kot pri MyBatisu. Hibernate precej pohitri razvojni čas – iz javanske kode lahko sestavljamo podatkovno bazo. Enostavni klici na podatkovno bazo so hitrejši in enostavneje sestavljeni kot pa v MyBatisu. Glede na enostavnost samega orodja je Hibernate z MyBatisom povsem primerljiv, vendar ga pri številu funkcionalnostih prekaša. Moje mnenje je, da je Hibernate, z vsemi svojimi funkcionalnostmi pri razvoju aplikacij, ki se izvajajo na strežniku in ni pričakovano veliko število klicev metod, ki hkrati dostopajo do istih podatkov na podatkovni bazi, izredno koristno in tudi hitro delujoče orodje. Pri aplikacijah, kjer pa je potrebno skrbeti tudi na prostor pomnilnika pri poganjanju aplikacij, pa je MyBatis še vedno izbira, za katero bi se odločil, saj pri prvi povezavi zavzame manj pomnilniškega prostora.

DataNucleus porabi še več časa za povezovanje z bazo kot pa Hibernate, od MyBatisa pa je približno štirikrat počasnejši. Za samo uporabo je bolj zapleten kot MyBatis, poizvedbe na podatkovno bazo pa so počasnejše. Samo količino kode in pa število nastavitvenih datotek nam zmanjša. Pri razvoju aplikacij, ki se izvajajo na strežniku in ni pričakovano veliko število klicev metod, ki hkrati dostopajo do istih podatkov na podatkovni bazi je, po mojem mnenju, zaradi večjega števila funkcionalnosti boljša izbira kot MyBatis. Omogoča nam tudi enostavno menjavanje podatkovnih virov – iz relacijske podatkovne baze na XML datoteko, na excelovo datoteko in podobno. Toda pri velikosti pomnilnika, ki ga zasede, je zelo zahteven. To pa nam pri večjih aplikacijah, kjer je potrebno skrbeti tudi na prostor pomnilnika z večimi hkratnimi dostopi do funkcij v aplikacij, lahko povzroči še velike težave.

V podjetju, kjer delam, se ukvarjamo z velikimi aplikacijami, katerih funkcije hkrati pokliče veliko število uporabnikov. Pri aplikacijah take vrste moramo biti pozorni na količino prostega pomnilnika na strežnikih, kjer aplikacije tečejo, zato je za podjetje najbolj smotrna uporaba orodja MyBatis. Ostali dve orodji prideta do izraza v aplikacijah, kjer je hkratnih

klicev funkcij, ki potekajo nad istimi podatki, in s tem ni veliko težav z zasedenostjo pomnilnika, manj.

6 Viri

[1] Java Persistence/Mapping. Zadnji dostop: 16.9.2011. Dostopno na: http://www.datanucleus.org/products/accessplatform/guides/jdo/tutorial.html

[2]MyBatis 3 – User guide, bela knjiga. Dostopno na: http://mybatis.googlecode.com/files/MyBatis-3-User-Guide.pdf

[3] Documentation - Hibernate - JBoss Community. Dostopno na:

http://www.hibernate.org/docs

[4] Using Java Data Objects - O'Reilly Media. Dotopno na:

http://www.onjava.com/pub/a/onjava/2002/02/06/jdo1.html?page=2

[5] Introduction to Java Data Objects (JDO) - Developer.com. Dostopno na:

http://www.developer.com/db/article.php/10920_1580231_2/Introduction-to-Java-Data-Objects-JDO.htm

[6] Slovar informatike. Dostopno na:

http://www.islovar.org

[7] Java brains – Hibernate. Dostopno na:

http://javabrains.koushik.org/p/hibernate.html

[8] PostgreSQL: The world's most advanced open source database. Dostopno na:

http://www.postgresql.org/

[9] Java Data Objects (JDO). Dostopno na: http://db.apache.org/jdo/

[10] Reflection (computer programming) - Wikipedia, the free encyclopedia. Dostopno na:

http://en.wikipedia.org/wiki/Reflection_(computer_programming)

[11] What is relational database. Dostopno na:

http://searchsqlserver.techtarget.com/definition/relational-database

[12] Object (Java 2 Platform SE 5.0). Dostopno na:

http://download.oracle.com/javase/1.5.0/docs/api/java/lang/Object.html

[13] java - How does JPA actually works? - Stack Overflow. Dostopno na: http://stackoverflow.com/questions/6301767/how-does-jpa-actually-works

[14] About reflection in Java/JPA database. Dostopno na:

http://www.objectdb.com/object/database/reflection

[15] Java programming dynamics, Part 2: Introducing reflection. Dostopno na:

http://www.ibm.com/developerworks/library/j-dyn0603/

[16] Byte Code Enhancement vs. Class Mutilation. Dostopno na: http://www.oreillynet.com/cs/user/view/cs_msg/12128

[17] DataNucleus 3.0 vs Hibernate 3.5 – Java Code Geeks. Dostopno na:

http://www.javacodegeeks.com/2011/02/datanucleus-30-vs-hibernate-35.html

[18] DataNucleus Access Platform – Guides. Dostopno na:

http://www.datanucleus.org/products/accessplatform/guides/index.html