TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia Teaduskond
Informaatikainstituut
Infosüsteemide õppetool

Java veebirakenduse reaktsiooniaja optimeerimine kasutades vahemälu

Bakalaureusetöö

Üliõpilane: Henri Liiv
Üliõpilaskood: 103995IAPB
Juhendaja: Raul Liivrand

Tallinn
2015

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

(*kuupäev*) (*allkiri*)

Annotatsioon

Java veebirakenduse reaktsiooniaja optimeerimine kasutades vahemälu.

Töö eesmärk on uurida Java veebirakenduse andmebaasiühenduse kihti, muutes korduvate andmete küsimine otstarbekamaks, kasutades ORM tarkvara Hibernate ja täpsemalt selle pakutavaid vahemälu (cache) komponente.

Töös uuritakse, millistel juhtudel on vahemälu kasutamine otstarbekas ja kui palju suudab selle kasutamine rakenduse tööd kiiremaks muuta. Töö annab ka lisaks hea ülevaate kuidas kasutada Hibernate vahemälu võimalusi.

Uurimuse jaoks luuakse Java veebirakendus kasutades Spring MVC raamistikku, andmebaasisüsteemina on kasutusel PostgreSQL, kuhu on genereeritud juhuslikud testandmed.

Töös käsitletakse probleeme, mis tekivad seoses vahemälu valesti konfigureerimisega. Näidatakse ja antakse soovitusi, kuidas vahemälu õigesti konfigureerida.

Töö tulemuseks on õigesti konfigureeritud Java veebirakendus, kus vahemälu kasutamine annab päringute puhul ajalise võidu. Töö käigus tehakse katseid, mis näitavad, kas vahemälu kasutamine on otstarbekas.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 45 leheküljel, 6 peatükki, 17 joonist, 6 tabelit.

Abstract

Java web application response time optimization using cache.

The purpose of this thesis is to study the Java web application's database connection layer. The aim is to find out how to make repeated data requests more rational. For that, Hibernate ORM is used with it's cache components.

Appropriate use of Hibernate cache is studied in this thesis as well as how much can caching make application to work faster. The thesis gives a good overview on how to use different features of Hibernate cache.

A Java web application is created for this thesis using Spring MVC framework. PostgreSQL is used as a database engine. Random test data is generated for the application.

Issues which may arise in connection with wrong configuration of the cache are studied in the thesis. Recommendations and right options are displayed on how to configure cache in the correct way.

The result of this thesis is a correctly configured Java web application with a working cache component. Tests are carried out to show whether using a cache is efficient.

The thesis is in estonian and contains 45 pages of text, 6 chapters, 17 figures and 6 tables.

Lühendite ja mõistete sõnastik

ORM	***Object Relational Mapping*** Tarkvara, mis teisendab Relatsioonilise andmebaasi tabelid objektide kujule – mida veebirakendus kasutab.
SQL	***Structured Query Language*** Päringukeel, mida toetavad kõik klient-server keskkonnale projekteeritud relatsioonandmebaasid.
IDE	***Integrated Development Environment*** Integreeritud arenduskeskond.
HQL	***Hibernate Query Language*** Hibernate poolt kasutatav päringukeel.
JMX	***Java Management Extensions*** Java tehnoloogia rakenduste monitooringuks.
HTTP	***Hypertext Transfer Protocol*** Hüperteksti edastusprotokoll.
MVC	***Model–view–controller*** Mudel, vaade, kontroller - tarkvara arhidektuuri mudel.
JSTL	***JavaServer Pages Standard Tag Library*** Java EE veebirakenduse arendusplatvorm.

Jooniste nimekiri

Tabelite nimekiri

Sisukord

1. Sissejuhatus

Vahemälu on mälu sagedasti kasutatavate andmete hoidmiseks. Vahemälu kasutatakse eeldusel, et andmete lugemine vahemälust toimub alati märksa kiiremini kui nende alalisest asukohast [1]. Konkreetse töö puhul kasutavad vahemälu andmebaasisüsteem (PostgreSQL) kui ka veebirakenduse ORM tarkvara (Hibernate).

Uurimustöös kasutatakse Hibernate ORM tarkvara, kuna tegu on selles valdkonnas kõige kasutatavama Java teegiga [2]. Hibernate teek pakub laialdasi võimalusi vahemälu realiseerimiseks, lubades selleks valida cache'i teegi. Selle töö puhul kasutatakse Ehcache teeki.

Hibernate vahemälu koosneb kolmest komponendist:

- Esimese taseme vahemälu (First Level Cache)
- Teise taseme vahemälu (Second Level Cache)
- Päringute vahemälu (Query Cache)

Selle töö käigus uuritakse:

- kuidas oleks võimalik muuta Java veebirakenduse reaktsiooniaega kiiremaks, kasutades selle realiseerimiseks vahemälu.
- Kuidas on otstarbekas kasutada vahemälu rakenduses, et säiliks andmete värskus.
- Uuritakse, milliste päringute puhul on vahemälu kasutamine mõistlik ja millise ajalise võidu annab selle kasutamine.
- Lisaks uuritakse, kuidas tuleks vahemälu seadistada antud süsteemi jaoks ja millist mõju avaldab suurem koormus, suurem kasutajate hulk, rakenduse reaktsiooniajale.

1.1 Taust ja probleem

Töö annab ülevaate vahemälu kasutamisest Java veebirakenduses, et muuta päringud andmekihi poole kasulikumaks, lugedes korduvalt küsitavaid andmeid sessioonipõhisest vahemälust.

Töö sisu annab veebiarendajale võimaluse tutvuda vahemälu kasutamise positiivsete ja negatiivsete külgedega. Samas annab ka ülevaate, kuidas oleks kõige mõistlikum vahemälu rakenduses kasutada.

Loodava tarkvara kvaliteedi mittefunktsionaalsetes nõuetes on enamasti tingimus, et tarkvara reaktsiooniaeg peab olema piisavalt lühike [3]. See tagab kliendi rahulolu tarkvaraga ja tähendab seda, et klient on teenusega rahul.

1.2 Ülesande püstitus

Eesmärgid, milleni töö käigus jõutakse:

- Tehakse ülevaade Hibernate vahemälu õigest kasutamisest.
- Uuritakse vahemälu käitumist erinevate päringute ja koormuste rakendamisel Java veebirakendusele.

1.3 Metoodika

Testmasinas käivitatakse Spring MVC veebirakendus integreeritud koos Hibernate teegiga. Veebirakendusel aktiveeritakse teise taseme vahemälu kasutamine ja päringu taseme vahemälu kasutamine.

Seejärel käivitatakse rakenduses erinevaid päringuid, mis testivad esmalt väikse koormuse juures vahemälu kasutamist võrdluses selle mittekasutamisega.

Peale seda kasutatakse rakendusele koormuse tekitamiseks tarkvara Apache JMeter, et uurida, kuidas käitub vahemälu suurema koormuse juures.

1.4 Ülevaade tööst

- Esimeseks tehakse sissejuhatus vahemälu kasutamisele Hibernate ORM'i puhul. Antakse ülevaade, kuidas toimib Hibernate kasutamisel vahemälu. Lisaks tutvustatakse vahendeid, millega uurimustulemusi saadakse ja töödeldakse.

- Järgmisena tutvustatakse rakenduse jaoks loodud PostgreSQL andmebaasi. Kirjeldatakse olemi-suhte diagrammi ja näidatakse tabelite täitumist.

- Edasi kirjeldatakse täpsemalt Hibernate vahemälu õigesti üles seadmist. Tuuakse välja põhilised situatsioonid, millele tasuks tähelepanu pöörata vahemälu kasutamisel veebirakenduse juures.

- Viimasena uuritakse vahemälu kasutamise ja mittekasutamise erinevusi – koormusega ja ilma. Tabelites ja graafikutel tuuakse visuaalselt välja uurimuste tulemused.

2. Vahemälu kasutamine veebirakenduses

Vahemälu kasutamise vajaduse tähtsus seisneb rakenduse jõudluse optimiseerimises. Vahemälu on komponent, mis paikneb rakenduse ja andmebaasi vahel. Selle eesmärgiks on ära hoida võimalikult paljude andmebaasipäringute tegemist, et muuta rakenduse reageerimisaega kiiremaks. See on eriti oluline kiiret reaktsiooniaega nõudvate rakenduste puhul.

2.1 Hibernate

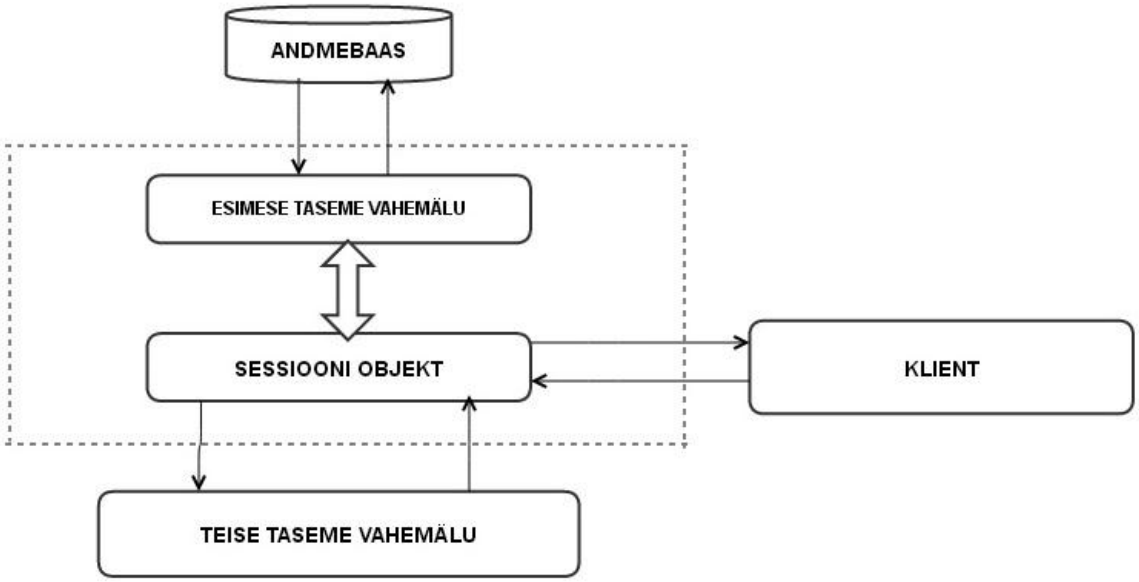
Hibernate ORM on vabavaraline teek, mis pakub võimalust siduda relatsioonilise andmebaasi skeem objektorienteeritud domeeniga. Esmaseks funktsiooniks on andmebaasi tabelite kaardistamine Java klassidesse. Hibernate pakub võimalust sõltumata andmebaasimoori tüübist, teha rakendusel päringuid andmebaasi poole. Hibernatel on olemas funktsioonid tegemaks andmebaasi päringuid ning pakub võimalust selle töö käigus uuritava vahemälu kasutamiseks.

2.2 Hibernate vahemälu

Hibernate pakub võimalust andmete säilitamiseks vahemälus. Kasutusel on kahetasemiline vahemälu.

Nagu on näha joonisel 1, siis Hibernate vahemälu jaguneb kaheks:

- Esimese taseme vahemälu
- Teise taseme vahemälu



Joonis 1. Hibernate vahemälu diagramm. [4]

2.2.1 Esimese taseme vahemälu (First level cache)

Esimese taseme vahemälu seostatakse alati sessiooni objektiga. Hibernate kasutab esimese taseme vahemälu vaikimisi ja selle kasutamist ei ole võimalik dikteerida. See töötleb ühe tehingu (transaktsiooni) teise järel, see tähendab, et ühte tehingut ei viida läbi mitu korda. Peamiselt vähendab see SQL päringute arvu, mida on vaja teha ühe tehingu käigus. Näiteks, selle asemel, et uuendada andmeid peale iga modifikatsiooni tehingu käigus, uuendab ta seda ainult ühe korra tehingu lõpus. [5]

2.2.2 Teise taseme vahemälu (Second level cache)

Lisaks esimese taseme vahemälule pakub Hibernate teek lisavõimalusi rakenduse optimeerimiseks, seda läbi teise taseme vahemälu.

Teise taseme vahemälu on seotud session factory objektiga. Tehingute ajal laetakse session factory tasemel teise taseme vahemälusse objektid. Need objektid on saadaval terve rakenduse kasutajate hulgale – ei piirduta vaid ühe kasutajaga. Kui päritav objekt on juba olemas vahemälus, siis leidakse see sealt ja tagastatakse meetodi väljakutsujale, see välistab ühenduse loomise andmebaasiga.

Teise taseme vahemälu põhjal töötab ka Hibernate poolt pakutav päringu vahemälu (Query Cache) , mida samuti antud selles töös uuritakse.

2.2.3 Päringu vahemälu (Query cache)

Hibernate pakub välja võimalust, kuidas säilitada päringute andmeid, kasutades selleks teise taseme vahemäluga tihedalt integreeritud päringu taseme vahemälu.

Nagu teise taseme vahemälu, on päringu taseme vahemälu kasutamine valikuline ning selle kasutamine tuleb vastavalt konfiguratsioonis seadistada.

Selle vahemälu puhul kasutatakse kahte salvestuspiirkonda. Üht, mis hoiab endas päringu tulemusi, ning teist, mis hoiab ajatempleid, mis näitavad, millal päritavaid andmeid viimati muudeti.

2.2.4 Vahemälu teenusepakkuja

Teise taseme vahemälu kasutamiseks tuleb valida sellele teenusepakkuja, mis on toetatav Hibernate poolt.

Hibernate pakub välja mitu erinevat teenusepakkujat, millel on realiseeritud valik erinevad vahemälu varundusstrateegiad (Cache Concurrency Strategy Support). Tabelis 1 tuuakse välja Hibernate’i poolt soovitatavad teenusepakkujad.

Vahemälu / varundusstrateegia	*Read-only*	*Nonstrict-read-write*	*Read-write*	*transactional*
Hashtable	Jah	Jah	Jah	
EHCache	Jah	Jah	Jah	
OSCache	Jah	Jah	Jah	
SwarmCache	Jah	Jah		
JBoss Cache 1.x	Jah			Jah
JBoss Cache 2	Jah			Jah

Tabel 1. Hibernate poolt soovitatavad vahemälu teenusepakkujad. [6]

Varundusstrateegia valik sõltub sellest, kas andmeid muudetakse või mitte.

Ülevaade vahemälu varundusstrateegiatest:

- **Read-only** strateegiat kasutatakse juhul, kui andmeid on vaja ainult lugeda, need ei ole muutmiseks. Kõige lihtsam ja kiiremini töötav strateegia.
- **Nonstrict read/write** on strateegia, mis ei lukusta kunagi vahemälu ja lubab andmete muutmist. Soovitatav kasutada juhul, kui andmeid muudetakse harva.
- **Read/write** on strateegia juhuks, kui andmeid on vaja muuta. Seda strateegiat ei ole soovitav kasutada, kui soovitakse kasutada **serializable** transaktsiooni isolaatsoni taset.
- **Transactional** on strateegia, mis tagab täielikult transaktsionaalse (fully transactional) vahemälu. [7]

Antud töös kasutatakse Ehcache vahemälu teeki.

2.3 Rakenduse töö monitoorimine

Veebirakenduse käitumise uurimiseks kasutatakse erinevaid rakendusi, mille abil saadakse töö jaoks vajalikke uurimustulemusi.

Olulisteks andmeteks on Hibernate päringute ja vahemälu statistika kuvamine. Hibernate puhul kasutakse JMX bean'i, et saada kätte rakenduse andmed. Vajalikke andmeid saadakse ka veebirakenduse konsoolist, kasutades IDE't.

2.3.1 JMeter

Apache JMeter [8] on graafiline Java veebirakenduse monitoorimise tööriist, mille abil saab rakenduse peal käivitada erinvevaid teste. Konkreetses töös kasutatakse Apache JMeter'it selleks, et läbi viia rakenduse koormusteste.

Kutsutakse välja katse jaoks vajalik funktsioon, kasutades selleks HTTP request'i. Tulemused salvestatakse ja nendest tehakse järeldused, kuidas töötab rakendus suurema hulga kasutajate puhul.

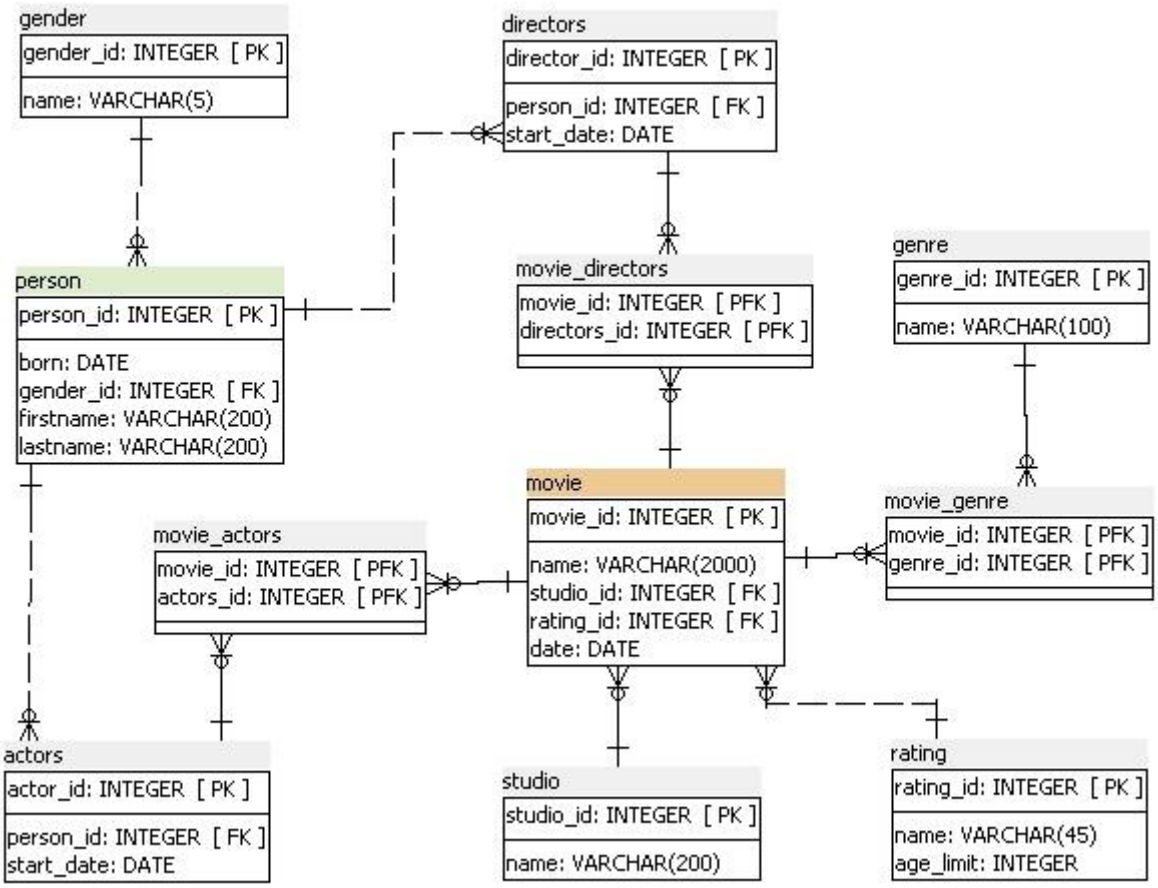
2.3.2 Jconsole

Jconsole graafiline kasutajaliides on monitoorimise vahend, mis vastab JMX spetsifikatsioonile. JConsole kasutab Java virtuaalmasinat, et anda teavet tulemuslikkuse ja ressursside tarbimise kohta töötavatel Java rakendustel.

Konkreetses töös kasutatakse rakenduse monitoorimiseks Jconsole'i koos Hibernate plugin'iga [9], mis annab ülevaate päringutest, vahemälu kasutamise efektiivsusest ja päringute ajalisest kestvusest. Saadav statistika annab aimu vahemälu kasutamise vajalikkusest.

3. Andmebaas

Andmebaasis hoitakse andmeid filmide, stuudiote, režissööride, näitlejate ja filmikategooriate kohta. Joonisel 2 esitatakse andmebaasi struktuuri kirjeldav diagramm (olemi-suhte diagramm on loodud kasutades vabavaralist tarkvara SQL Power Architect). Kasutatud on erinevaid seosetüüpe, et testida võimalikult täpselt ja laialdaselt vahemälu kasutamist. Andmebaasi süsteemina kasutatakse PostgreSQL 9.4.



Joonis 2. PostgreSQL andmebaasi tabelite struktuur.

3.1 Andmed

Tegu on analüüsiks kasutatava andmebaasiga, mille kirjetel puudub praktiline tähendus, see tähendab, et testandmed said genereeritud juhuslikkuse alusel, kasutades selleks PHP skripti.

Tabelites on lisatud adekvaatsemate tulemuste saamiseks erinevaid kirjeid suuremates kogustes. Tabelis 2 on välja toodud tabelite ridade kogused terve andmebaasi kohta.

Suurema hulga andmete korral on lihtne anda andmebaasile edasi päringuid, mille teostamine võtab kauem aega – hiljem on võimalik neid päringuid vahemälu kasutamisega kiiremaks muuta.

Tabeli nimi	***Kirjete kogus tabelis***	***Andmete kogumaht***
actors	19,998,460	1707MB
directors	19,998,460	1780MB
gender	2	24KB
Genre	82	24KB
Movie	37,881,899	6551MB
movie_actors	216,562	23MB
movie_directors	205,049	22MB
movie_genre	30,542,567	3198MB
person	39,996,921	4522MB
Rating	5	24KB
Studio	1,000	88KB

Tabel 1. Andmebaasiskeemi movies tabelite mahud.

4. Rakendus

Rakenduse implementeerimiseks kasutatake Java Spring MVC raamistikku, millega on kombineeritud Hibrernate teek, lisaks veel JSTL teek kasutajale andmete kuvamiseks. Spring MVC raamistik pakub võimalusi luua uusimaid tehnoloogiaid kasutades rakendus vahemälu kasutamise uurimiseks. Rakendus käivitatakse Apache Tomcat 7 serveri konteineris.

Töö jaoks kasutatakse riistvarana personaalarvutit (CPU: 4 x 2.40 GHz x86_64, muutmälu: 4 GB), millel antud töö jaoks vajalikud rakendused käivitatakse.

Kasutusel on Spring Framework versiooniga 4.1.1, Hibernate versiooniga 4.2.4. eelpool nimetatud teegid on paigaldatud kasutades Apache Maveni tarkvara manageerimise süsteemi.

IDE'na kasutatakse IntelliJ IDEA 14.1 keskkonda.

4.1 Andmebaasi tabelite kaardistamine rakenduse jaoks

Töö põhiliseks uuritavaks objektiks on rakenduse ja andmebaasi vaheline suhtlus ja selle optimeerimine. Sellepärast on olulisel kohal andmebaasi tabelite kaardistamine objektorienteeritud kujule. Töös kasutatav Hibernate teek pakub selleks võimalusi. Konifigureerimiseks kasutatakse Java annotatsioone nagu on näidatud joonisel 3. Õigesti kaardistatud rakendus saab oluliseks töö teises pooles, kui objektiklasside peal kasutatakse vahemälu annotatsioone.

```
@Entity
public class Movie {
    private int movieId;
    private String name;
    private Date date;
    private Studio studio;
    private Rating rating;
    private Set<Genre> genres;
    private Set<Actors> actors;
    private Set<Directors> directors;


@ManyToOne
@JoinColumn(name="rating_id")
public Rating getRating() {
```

```
    return rating;
}

public void setRating(Rating rating) {
    this.rating = rating;
}

@ManyToMany
@JoinTable(name ="movie_genre",
            joinColumns={@JoinColumn(name="movie_id")},
            inverseJoinColumns={@JoinColumn(name="genre_id")})
public Set<Genre> getGenres() {
    return genres;
}

public void setGenres(Set<Genre> genres) {
    this.genres = genres;
}
}
```

Joonis 3. Andmebaasi tabeli movie kaardistamine kasutades Java annotatsioone.

Joonisel 3 on näha Java klassifaili, millel defineeritakse klassi väljad vastavalt andmebaasi väljadele. Joonisel 3 on genereeritud väljadele rating ja genre getter ja setter meetodid. Nagu näha joonisel 2, viitavad antud väljad tabelitele rating või genre. Tabelite movie ja rating vahel on võimsussuhte tüübiks mitu-üks suhtetüüp sellepärast märgitakse vastav välja kohale annotatsioon **@ManyToOne** ning lisatakse sellele täpsustus, millise väljaga tabelis rating vastav tabel seotakse (@JoinColumn).

Sarnaselt käitutakse mitu-mitu võimsussuhte puhul. Väli või välja getter meetod tähistatakse annotatsiooniga **@ManyToMany** ja ühendatakse vahetulemuste tabeliga @JoinTable annotatsiooniga. @JoinTable annotatsiooni puhul peab tähele panema millised väljad vahetabeliga ühendatakse ja kaardistama vastavad väärtused.

4.2 Vahemälu kasutamine rakenduses

Nagu peatükis 2.2.1 kirjeldatud, kasutab Hiberante vaikimisi aktiveeritult esimese taseme vahemälu, teise taseme vahemälu ning päringu vahemälu sisse lülitamiseks on vajalik see Hibernate konfiguratsioonis aktiveerida. Joonisel 4 on toodud näide Hibernate

konfigureerimisest, kasutades selleks Spring MVC seadistuste faili mvc-dispatcher-servlet.xml.

Vahemälu kasutamine määratakse rakenduse session factory bean'is.

Hibernate konfiguratsioonis olulised omadused (properties) teise taseme vahemälu kasutamiseks, joonisel 4 on toodud täielik Hibernate konfiguratsioon**:**

- **hibernate.generate_statistics** – päringute ja vahemälu kasutamise statistika – näitab paljudel kordadel vahemälu poole pöörduti jms.
- **hibernate.cache.use_second_level_cache** – teise taseme vahemälu kasutamise sisselülitamine.
- **hibernate.cache.use_query_cache –** päringute vahemälu sisse lülitamine.
- **hibernate.cache.region.factory_class –** vahemälu teenusepakkuja valimine, antud rakendus kasutatakse sellena EhCache vahemälu.
- **net.sf.ehcache.configurationResourceName** - vahemälu konfiguratsioonifaili asukoht rakenduses.

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate4.LocalSessionFactoryBean">
    <property name="dataSource" ref="dataSource"/>
    <property name="packagesToScan" value="com.springapp.mvc.Model"/>
    <property name="hibernateProperties">
        <props>
            <prop
key="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</prop>
            <prop key="hibernate.generate_statistics">true</prop>
            <prop key="hibernate.cache.use_second_level_cache">true</prop>
            <prop key="hibernate.cache.use_query_cache">true</prop>
            <prop
key="hibernate.cache.region.factory_class">org.hibernate.cache.ehcache.EhCa
cheRegionFactory</prop>
            <prop key="net.sf.ehcache.configurationResourceName">/cache-
config.xml</prop>
        </props>
    </property>
</bean>
```

Joonis 4. Hibernate vahemälu konfigureerimine Spring MVC veebirakenduses.

Ehcache konfigureerimiseks kasutatakse ehcache-config.xml faili, millele viidatakse session factory bean'is. Joonisel 5 on kujutatud olulisemat osa XML-failist, milles määratakse vahemälusse salvestamise strateegiad.

Joonisel 5 on esitatud kahte XML elementi, millega vahemälu seadistatakse:

- **DefaultCache** element on vahemälu seadistus, mis määrab ära uue loodava vahemälu salvestusparameetrid. Selline seadistus antakse vahemälu objektidele, mis luuakse programmselt.

- **Cache** element määrab vahemälu strateegia kindlale objektile, joonisel 5 on selleks StandardQueryCache (määrab päringu vahemälu salvestuspiirkonna konfiguratsiooni).

```
<ehcache>
   <diskStore path="java.io.tmpdir"/>

   <defaultCache
         maxElementsInMemory="50000"
         maxElementsOnDisk="10000"
         eternal="false"
         timeToLiveSeconds="86400"
         overflowToDisk="false"
         memoryStoreEvictionPolicy="LRU"
         />


   <cache name="org.hibernate.cache.internal.StandardQueryCache"
         maxElementsInMemory="50000"
         maxElementsOnDisk="10000"
         eternal="false"
         timeToLiveSeconds="86400"
         overflowToDisk="false"
         memoryStoreEvictionPolicy="LRU"
         />
</ehcache>
```

Joonis 5. Ehcache konfiguratsioon.

Elementide sees määratakse ära vahemälu regiooni salvestusparameetrid, järgnevalt tuuakse tabelis 2 ülevaade olulisematest.

Attribuut	Väärtus	Seletus
maxElementsInMemory	number	Mälus hoitavate objektide arv.
maxElementsOnDisk	number	Mälukettal hoitavate objektide arv.
eternal	true/false	Määrab andmete hoiustamise vahemälus igaveseks, kirjutab üle timeToLive ja timeToIdle.
timeToLiveSeconds	aeg, millisekundites	Aeg,kui kaua hoitakse objekte mälus.
overflowToDisk	number	Kui attribuuti maxElementsInMemory ületatakse, salvestatakse objektid mälukettale.
memoryStoreEvictionPolicy	LRU (least recently used), LFU (least frequently used), FIFO (first in first out, vanim element loomise aja järgi).	Mälu vabastamise strateegia.

Tabel 2. Olulisemad cache elemendi väärtused vahemälu konfiguratsioonifailis.

Teise taseme vahemälu käivitamiseks tuleb täiendada mudelite klasse, lisades klassi märksõna ette koodis @Cache annotatsiooni. See näitab Hibernate'le, et antud objektiklassi andmeid soovitakse varundada vahemälus.

Nagu näha joonisel 6, siis kasutatakse @Cache annotatsiooni **mitu-mitu** võimsussuhtega vahetabeli juures.

Kuna rakenduses ei looda vahetabelite jaoks eraldi objektilist vastavust (vahetabeli objektiklassi), siis viidatakse tabelile movie_genre lisades @Cache annotatsiooni getter meetodile. Sedasi salvestatakse vahetabeli tulemused vahemälus.

@Cache annotatsiooni järele märgitakse vahemälu varundamise strateegia tüüp ja vahemälu regioon.

```
@Entity
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY, region =
"movie")
public class Movie {
```

```
    private int movieId;
    private String name;
    private Date date;
    private Studio studio;
    private Rating rating;
    private Set<Genre> genres;
    private Set<Actors> actors;
    private Set<Directors> directors;

@ManyToMany
@Cache(usage = CacheConcurrencyStrategy.READ_ONLY, region =
"movie_genre")
@JoinTable(name ="movie_genre",
            joinColumns={@JoinColumn(name="movie_id")},
            inverseJoinColumns={@JoinColumn(name="genre_id")})
public Set<Genre> getGenres() {
    return genres;
}
```

Joonis 6. Mudeli klassi vahemälu (@Cache) annotatsioonid.

4.3 Päringu vahemälu kasutamine (query cache)

Hibernate pakub võimalust lisaks eelnevalt esitletud võimalustele kasutada päringu vahemälu.

Päringu vahemälu ei varunda endas reaalseid olemeid – seal hoitakse identifikaatori väärtuseid ja tulemuse väärtuste tüüpe. Sellel põhjusel peaks kasutama päringu vahemälu alati koos teise taseme vahemäluga. [7]

Päringu vahemälu kasutamiseks tuleb see esmalt aktiveerida session factory bean'is nagu on kirjeldatud joonisel 4.

Seejärel tuleb kirjeldada programmikoodis, et antud päringu tulemused vahemälusse paigutatakse, nagu on kirjeldatud joonisel 7 (Query objektil määratakse väli cacheable tõeväärtus true, kasutades selleks meetodit setCacheable).

```
public List<Person> getAllPersons() {

    String sql = "select m from Movie m where m.movieId<500";
    Query query =
sessionFactory.getCurrentSession().createQuery(sql);
    query.setCacheable(true);

    return (List<Person>)query.list();
}
```

Joonis 7. Päringu vahemälu kasutamine - aktiveerimine programmikoodis.

5. Vahemälu kasutamise uurimine veebirakenduses

Vahemälu kasutamise uurimiseks viiakse läbi erinevaid katseid. Katsete tulemustest võib teha järelduse vahemälu kasutamise mõttekuse kohta.

5.1 Vahemälu kasutamine rakenduses

Esimese taseme vahemälu. Andmevahetus vahemäluga toimub ainult sessiooni siseselt. Esimese taseme vahemälu kasutamise testi kood on näha joonisel 8.

```
Session session = sessionFactory.openSession();
session.beginTransaction();
Movie movie = (Movie) session.get(Movie.class,1);
movie.setName("Avengers");

Movie movie2 = (Movie) session.get(Movie.class,1);
session.getTransaction().commit();
session.close();
```

Joonis 8. Esimese taseme vahemälu testi näitekood.

Konsooli väljund koodi täitmisel:

select movie0_.movie_id as movie_id1_4_2_, movie0_.date as date2_4_2_, movie0_.name as name3_4_2_, vie0_.rating_id as rating_i4_4_2_, movie0_.studio_id as studio_i5_4_2_, rating1_.rating_id as rating_i1_6_0_, rating1_.age_limit as age_limi2_6_0_, rating1_.name as name3_6_0_, studio2_.studio_id as studio_i1_7_1_, studio2_.name as name2_7_1_ from Movie movie0_ left outer join Rating rating1_ on movie0_.rating_id=rating1_.rating_id left outer join Studio studio2_ on movie0_.studio_id=studio2_.studio_id where movie0_.movie_id=1

update Movie set date='04/12/1949 00:00:00.000', name='Avangers', rating_id=4, studio_id=957 where movie_id=1

Kommentaar: Konsooli väljundist on näha, et ühe transaktsiooni käigus pöörtuti andmebaasi poole kaks korda. Esimesel korral küsiti andmeid antud filmi kohta ning seejärel muudeti

filmi nime. Teistkordsel andmete küsimisel enam päringuga andmebaasi poole ei pöördutud, vaid tulemus tagastati vahemälust.

Teise taseme vahemälu. Vahemälu tulemusi salvestatakse üle erinevate sessioonide. Teise taseme vahemälu testi kood on näha joonisel 9.

```
Session session = sessionFactory.openSession();
session.beginTransaction();
Movie movie = (Movie) session.get(Movie.class,1);
System.out.println(movie);
session.getTransaction().commit();
session.close();

Session session2 = sessionFactory.openSession();
session2.beginTransaction();
Movie movie2 = (Movie) session2.get(Movie.class,1);
System.out.println(movie2);
session2.getTransaction().commit();
session2.close();
```

Joonis 9. Teise taseme vahemälu testi näitekood.

Konsooli väljund koodi täitmisel (konsooli väljund)**:**

select movie0_.movie_id as movie_id1_4_2_, movie0_.date as date2_4_2_, movie0_.name as name3_4_2_, movie0_.rating_id as rating_i4_4_2_, movie0_.studio_id as studio_i5_4_2_, rating1_.rating_id as rating_i1_6_0_, rating1_.age_limit as age_limi2_6_0_, rating1_.name as name3_6_0_, studio2_.studio_id as studio_i1_7_1_, studio2_.name as name2_7_1_ from Movie movie0_ left outer join Rating rating1_ on movie0_.rating_id=rating1_.rating_id left outer join Studio studio2_ on movie0_.studio_id=studio2_.studio_id where movie0_.movie_id=1

Esimese päringu täitmise aeg: **1718ms**

Teise päringu täitmise aeg: **1ms**

Kommentaar: Konsooli väljundist on näha, et päringut täidetakse ainult ühe korra. Teise taseme vahemälu kasutamisel säilivad tulemused vahemälus kõikide sessioonide jaoks. Teise päringu tulemus loeti otse vahemälust ning see oli ajaliselt kiirem kui esimene.

5.2 Vahemälu tulemuste hoidmine arvuti mälus.

Päring (HQL)**:** SELECT m FROM Movie m WHERE m.movieId<4

Päringu vahemälu (StandardQueryCache) regioon pärast päringu käivitamist (konsooli väljund)**:**

key: sql: select movie0_.movie_id as movie_id1_4_, movie0_.date as date2_4_, movie0_.name as name3_4_, movie0_.rating_id as rating_i4_4_, movie0_.studio_id as studio_i5_4_ from Movie movie0_ where movie0_.movie_id<?; parameters: ; named parameters: {id=4}; transformer: org.hibernate.transform.CacheableResultTransformer@110f2 value: [5862581127536640, 1, 2, 3]

Teise taseme vahemälu pärast päringu käivitamist (konsooli väljund):

CacheEntry(com.springapp.mvc.Model.Movie)[1,1949-04-12 00:00:00.0,1,1,geck demonetized hardball scenics securitized Lombard epistles ,4,957]

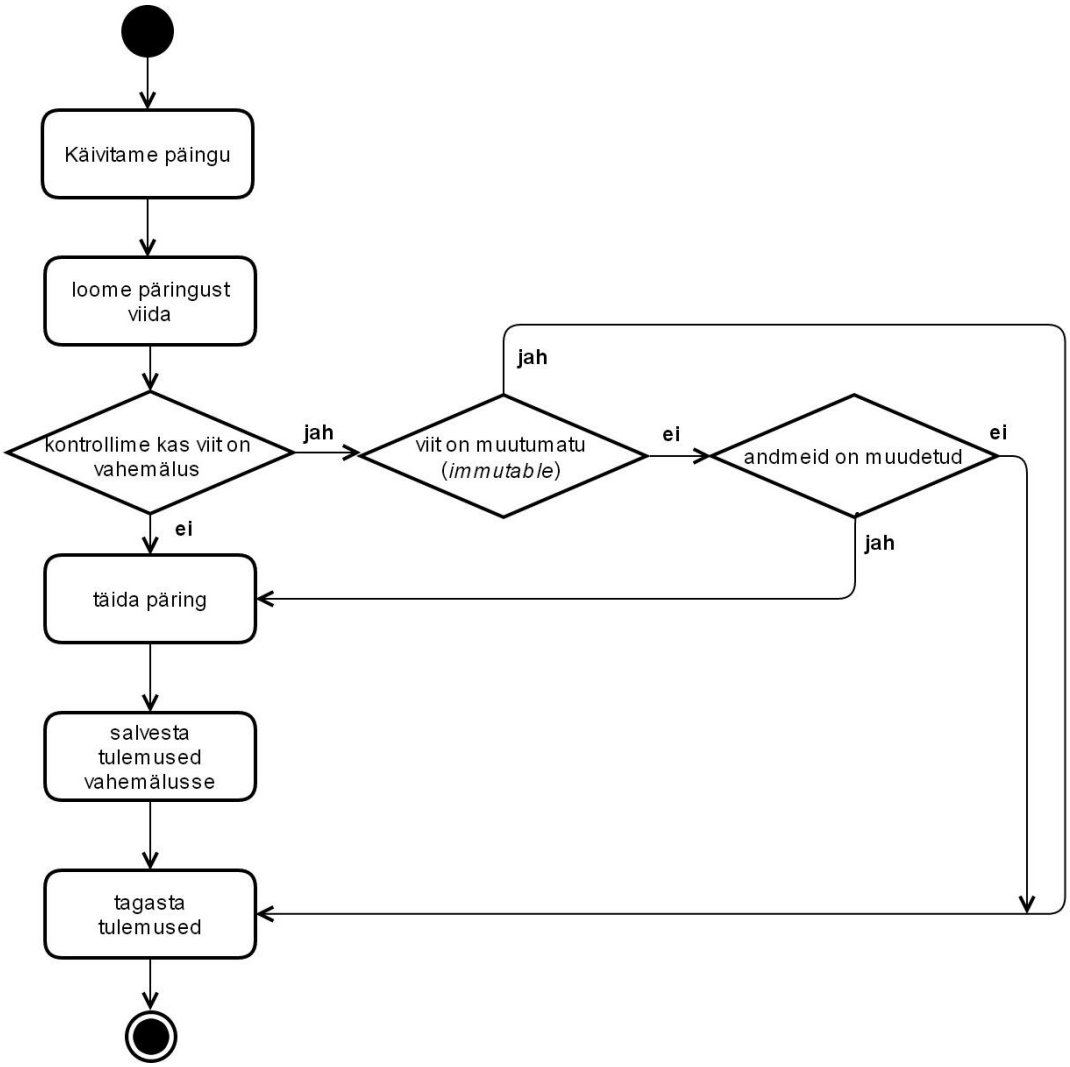
CacheEntry(com.springapp.mvc.Model.Movie)[2,1940-04-12 00:00:00.0,2,2,eight tardiness outcompeted lobotomizes penological analytically gaped ,5,353]

CacheEntry(com.springapp.mvc.Model.Movie)[3,1935-04-12 00:00:00.0,3,3,diammonium Kodiak barley-broth cicatrised healm midleg lookouts ,4,378]

Kommentaar:

Konsooli väljundist on näha, et päringu vahemälus hoitakse tulemuste viitasid või identifikaatoreid. Võtmeks on sel juhul päring ise ja sellele antud parameetrid, tulemuste värskust silmaspidades salvestatakse lisaks timestamp ehk ajatempel. Joonisel 10 on näha, kuidas toimib HQL päring, kui kasutatakse vahemälu.

Teise taseme vahemälus hoitakse andmeid terve olemi kohta, viited teistes tabelites olevatele andmetele on esitatud identifikaatori kujul.



Joonis 10. Hibernate päringu täitmine, kui kasutusel on teise taseme vahemälu. [10]

5.3 Tavapäringu ja vahemälu pöördumise ajaline võrdlus

Selles peatükis näidatakse näidatakse, kas korduvate päringute puhul on võimalik saada ajaline võit, kui kasutada selleks Hibernate vahemälu pakutavaid võimalusi.

Päringud käivitatakse esmalt otse andmebaasisüsteemis SQL kujul, seejärel tehakse päring veebirakenduses läbi Hibernate ning viimasena kasutatakse Hibernate juures teise taseme ja päringu vahemälu. Viimase katse (Hibernate koos vahemäluga) aeg on saadud tingimusel, et päring on juba varundatud päringu vahemälus ja olemid on teise taseme vahemälus salvestatud.

Päring: SELECT movie_id, date, name, rating_id, studio_id FROM Movie WHERE movie_id<100;

Päringu väljakutsumise asukoht	**Päringu kestvus (ms)**
pgAdmin III	**15**
Hibernate ilma vahemälu kasutamata	**169**
Hibernate koos vahemäluga	**76**

Tabel 3. Tavapäringu ja vahemälu pöördumise ajaline võrdlus - lihtne päring.

Päring (HQL)**:** SELECT p FROM Person p
JOIN p.gender ge
JOIN p.directors d
JOIN d.movies m
JOIN m.studio s
JOIN m.genres g
JOIN m.rating r
WHERE p.firstname = 'Aaren Aron' AND ge.name='M'

Päringu väljakutsumise asukoht	**Päringu kestvus (ms)**
pgAdmin III	**290226**
Hibernate ilma vahemälu kasutamata	**291910**
Hibernate koos vahemäluga	**69**

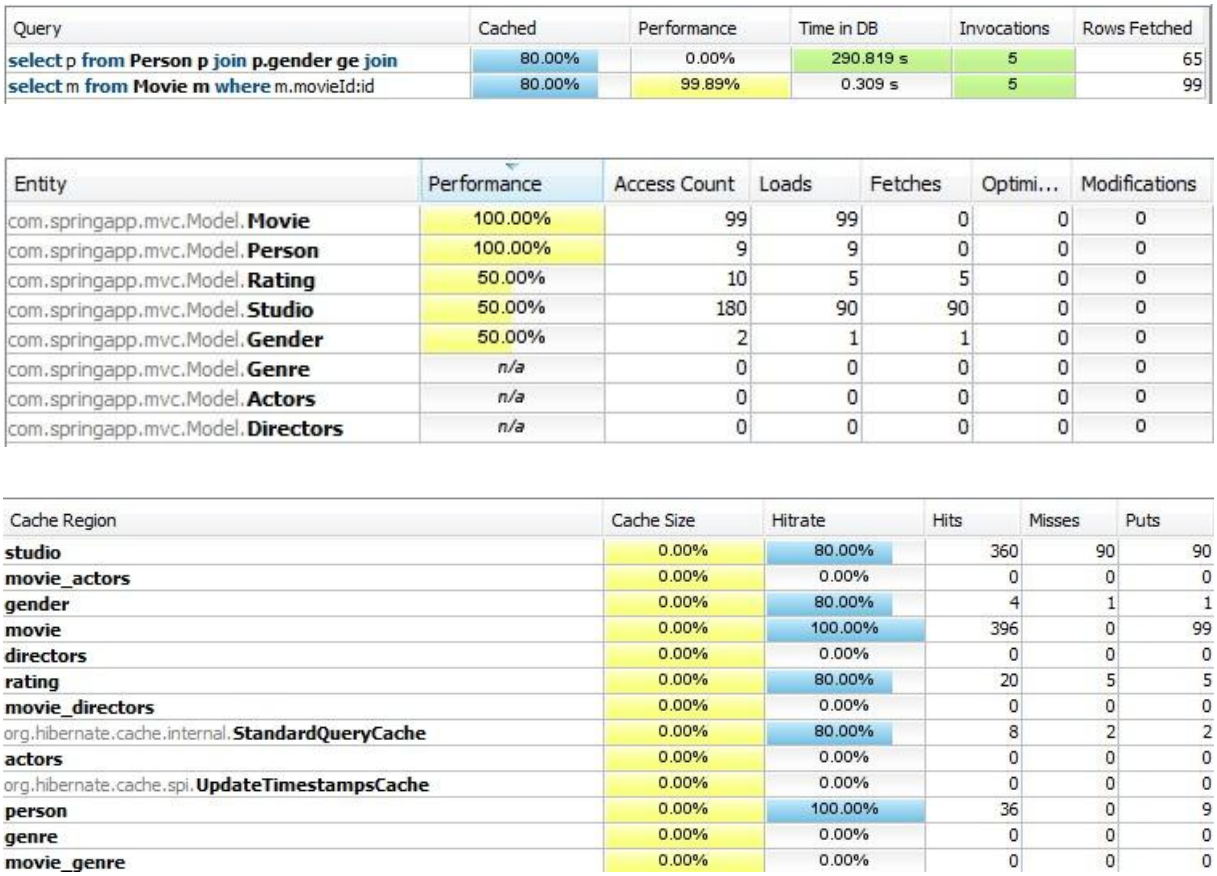
Tabel 4. Tavapäringu ja vahemälu pöördumise ajaline võrdlus - teksiotsing.

Esimese päringu käivitamine erinevatelt algtingimustelt näitas, et vahemälu kasutamine rakenduse kiiruse tõstmiseks tähendas ajalist võitu.

Teise päringu käivitamine näitas, et suure hulga andmete töötlemine testarvutis on aeganõudev. Peale päringu tulemuste vahemälusse laadimist kahanes andmete kättesaamise aeg sarnasele tasemele esimese päringuga.

Võib järeldada, et vahemälust tulemuste tagastamine on mõlema päringu puhul ajaliselt võrdne. Selle kiirus sõltub sellest, kui palju antud olem arvuti ressursse kasutab - uusi andmebaasi päringuid selle käigus läbi ei viida. Nagu eelpool kirjeldatud - vahemälu andmetele pääsetakse ligi läbi identifikaatorite.

Jooniselt 11 on näha, kuidas 5 identse päringu puhul vahemälus hoitavaid olemeid tabatakse ja milline on päringu efektiivsus, kui kasutatakse vahemälu. Võib näha, et mõlema päringu puhul on Hibernate pöördunud esmalt andmebaasi poole ning seejärel andmed arvuti mälusse salvestatud. Edasiste käivitamiste puhul on loetud tulemused otse vahemälust.



Query	Cached	Performance	Time in DB	Invocations	Rows Fetched
select p from Person p join p.gender ge join	80.00%	0.00%	290.819 s	5	65
select m from Movie m where m.movieId:id	80.00%	99.89%	0.309 s	5	99

Entity	Performance	Access Count	Loads	Fetches	Optimi...	Modifications
com.springapp.mvc.Model.**Movie**	100.00%	99	99	0	0	0
com.springapp.mvc.Model.**Person**	100.00%	9	9	0	0	0
com.springapp.mvc.Model.**Rating**	50.00%	10	5	5	0	0
com.springapp.mvc.Model.**Studio**	50.00%	180	90	90	0	0
com.springapp.mvc.Model.**Gender**	50.00%	2	1	1	0	0
com.springapp.mvc.Model.**Genre**	*n/a*	0	0	0	0	0
com.springapp.mvc.Model.**Actors**	*n/a*	0	0	0	0	0
com.springapp.mvc.Model.**Directors**	*n/a*	0	0	0	0	0

Cache Region	Cache Size	Hitrate	Hits	Misses	Puts
studio	0.00%	80.00%	360	90	90
movie_actors	0.00%	0.00%	0	0	0
gender	0.00%	80.00%	4	1	1
movie	0.00%	100.00%	396	0	99
directors	0.00%	0.00%	0	0	0
rating	0.00%	80.00%	20	5	5
movie_directors	0.00%	0.00%	0	0	0
org.hibernate.cache.internal.**StandardQueryCache**	0.00%	80.00%	8	2	2
actors	0.00%	0.00%	0	0	0
org.hibernate.cache.spi.**UpdateTimestampsCache**	0.00%	0.00%	0	0	0
person	0.00%	100.00%	36	0	9
genre	0.00%	0.00%	0	0	0
movie_genre	0.00%	0.00%	0	0	0

Joonis 11. JConsole-Hibernate testpäringute vahemälu statistika.

5.4 Tavapäringu ja vahemälu pöördumise ajaline võrdlus kui rakendusele langeb suurem koormus.

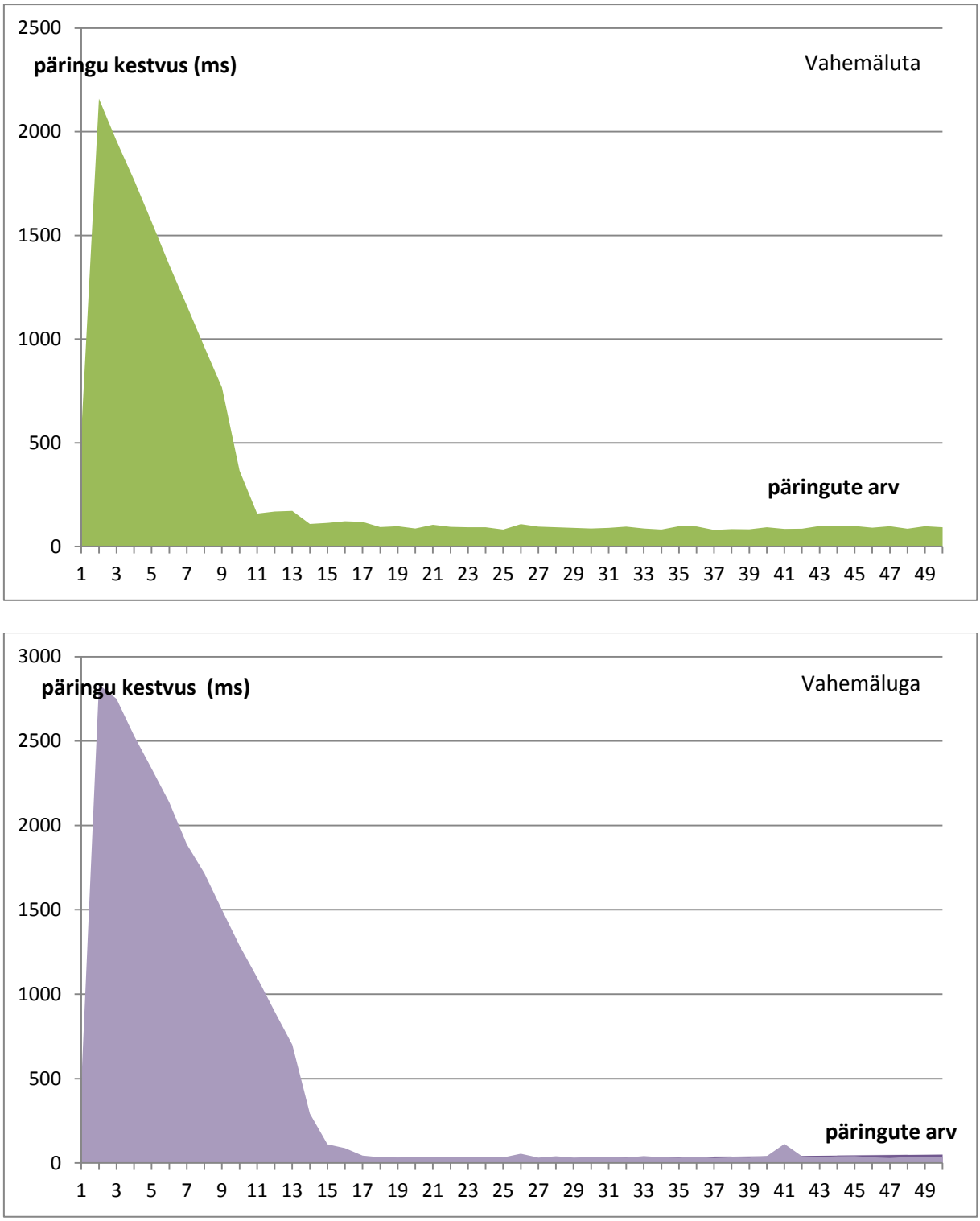
Loome rakendust JMeter kasutades testjuhu, mis simuleeriks rakendusele koormust. Võrdleme tulemusi juhtudel, kui kasutatakse vahemälu ja kui vahemälu ei kasutata. Andmeid päritakse läbi veebirakenduse, kasutades HTTP request'i.

Katse 1:

Lõimede arv (kasutajate arv): **50**

Päringute edastamise periood: **10** s

Päring (HQL): SELECT movie_id, date, name, rating_id, studio_id FROM Movie WHERE movie_id<=100;



Joonis 12. Päringute kestvus suurema koormuse puhul - vahemälu kastuamata.

Ilma vahemäluta päringu keskmine kestvus	332 ms
Vahemäluga päringu keskmine kestvus	479ms

Tabel 5. Päringute ajaline kestvus – aritmeetiline keskmine

Joonisel 12 on näidatud lihtsama sisuga päringu, millega küsitakse identifikaatori järgi 100 esimest filmi, andmebaasi poole pöördumiste ja tulemuste tagastamise ajad.

Tulemustest on näha, et puuduvad suuremad erinevused vahemälu kasutamise ja selle mittekasutamise vahel. Keskmise päringu kestvuse võrdluses on näha, et sellise päringu tegemine võtab oodatust kauem aega, võrreldes vahemälu mittekasutamisega.

Pärast esimese päringu käivitamist laetakse päringu tulemused andmebaasi enda kohalikku vahemälu puhvrisse. Selle tulemusel päringu aeg langeb tunduvalt ja jääb püsima umbes 100 millisekundi juurde.

Vahemälu kasutades kantakse pärast esimese päringu lõpetamist tulemused teise taseme vahemälusse. Peale seda tagastatakse tulemused umbes 30 millisekundiga.

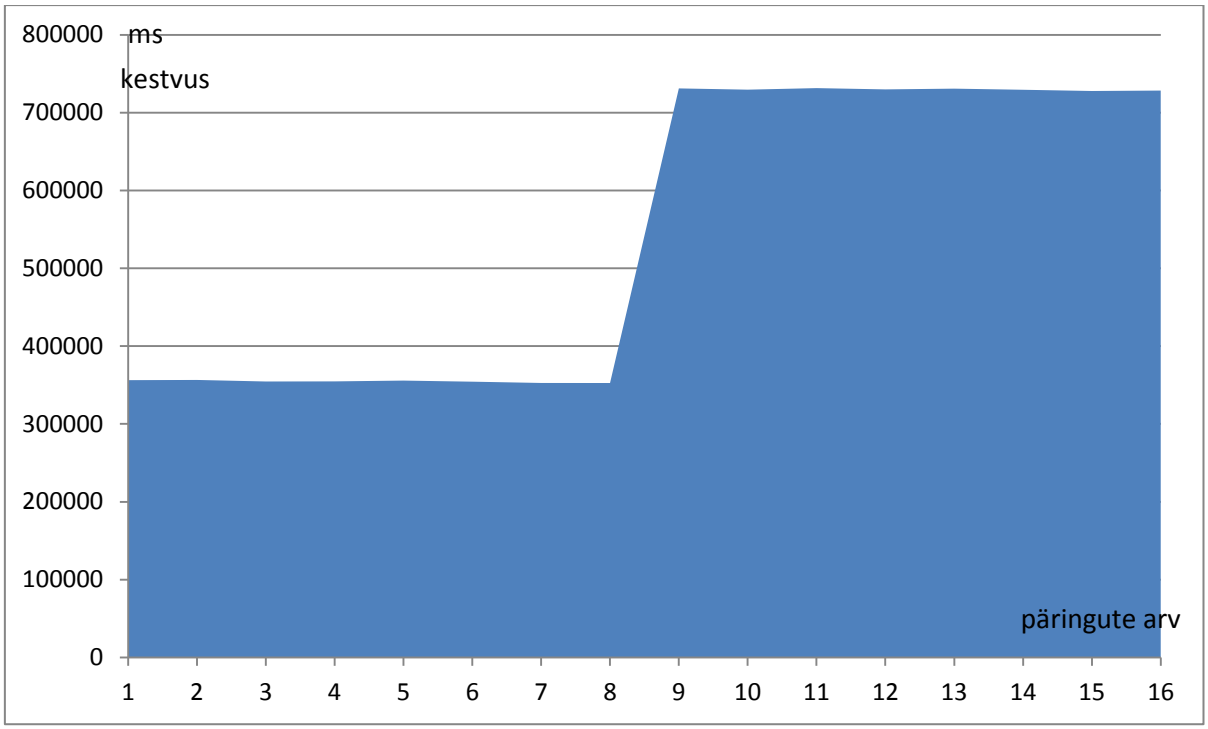
Saadud tulemustest võib järeldada, et lihtsa päringu puhul erilist võitu ei saavutata, samas on Hibernate vahemälu kasutamisel näha, et tulemused tagastatakse kiiremini võrreldes andmebaasi pöördumise stsenaariumiga. Hibernate vahemälu kasutamine, säästaks rakendust kulukast andmebaasipäringu saatmisest.

Katse 2:

Lõimede arv (kasutajate arv): **8**

Päringute edastamise periood: **16s**

Päring: SELECT p from Person p where p.firstname = "Aaren Aaron"



Joonis 13. Tekstiotsing tabelist person vahemälu kasutamata.

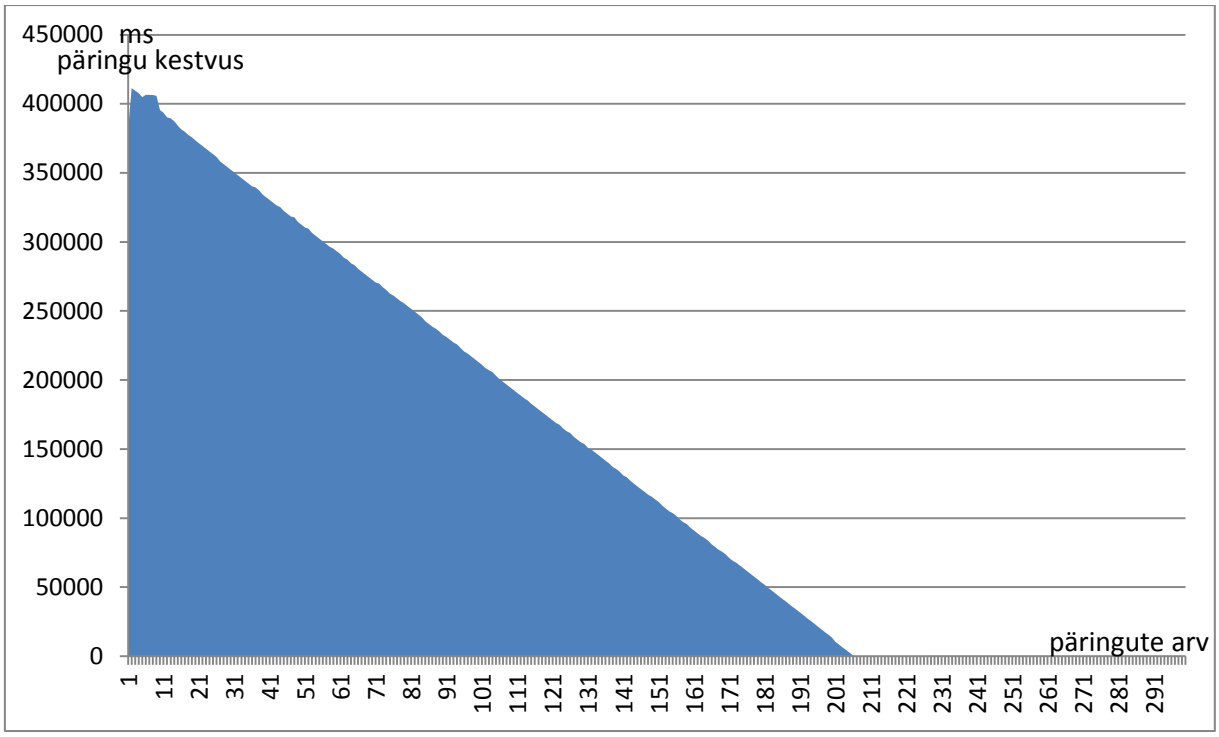
Nagu joonisel 13 on näha, siis tekstiotsingu puhul võtab päringu tulemuste saamine kauem aega kui "katse 1" päringus. Selline täitmisaeg on põhjustatud testmasina vähesest võimekusest, päringu keerukusest, täpsemalt suurema hulga andmete töötlemise vajadusest. Lisaks sellele ei hoita tekstiotsingu tulemusi postgreSQL'i puhvri mälus, mis teeks korduvad päringud andmebaasi poolelt kiiremaks.

Katse 3:

Lõimede arv (kasutajate arv): **300**

Päringute saatmise periood: **600s**

Päring: SELECT p from Person p where p.firstname = "Aaren Aaron"



Joonis 14. Päringute kestvus tekstiotsingul kasutades päringu vahemälu.

Kuna tekstiotsingu päring võttis andmebaasi poolt oodatust kauem aega, anti järgmises katses rakendusele päringuid pikema perioodi jooksul, et saada täpsem ülevaade.

Joonisel 12 on näha, et pärast esimeste päringute lõpetamist umbes 400 000 millisekundi jooksul kasutavad järgnevad päringud kasutama juba vahemälusse salvestatud andmeid.

Võib järeldada, et vahemälu kasutamine viimase päringu puhul annab antud olukorras märgatava ajalise võidu. Päringule tagastati vaste umbes 30 millisekundi jooksul.

Üldistest tulemustest võib järeldada, et väheste andmemahtude korral on Hibernate vahemälust tulemuste lugemine kiirem kui andmebaasi enda vahemälu puhvrist.

5.5 Suurema hulga päringutulemuste hoidmine veebirakenduse vahemälus

Ehcache pakub võimalust hoida tulemusi nii arvuti mälus kui ka kõvakettal. Sobilik on seadistada vahemälu regiooni suurus selliseks, nagu on vastavas andmebaasitabelis andmeid.

Katse 1:

Katse käigus küsiti andmebaasilt andmeid eesmärgiga, et tagastatavate andmete maht oleks suurem, kui on vahemälu regiooni konfiguratsioonis määratud. Joonisel 15 on näha, et vahemälu seadetes märgiti maksimaalseks elementide arvuks mälus 20000, päringuga tagastati aga elementide arv, mis oli suurem, kui maksimaalne kogus elemente.

Päring: SELECT p from Person p where p.firstname = "Aaren Aaron"

Antud juhul salvestati päring päringute vahemälu regiooni (StandardQueryCache). See tähendas, et Hibernate teadis, milliseid andmeid oli vaja andmebaasist küsida, sest päringu vahemälus on salvestatud tulemuste identifikaatorid. Identifikaatorite järgi päriti andmebaasilt iga objekt eraldi.

See muutis lõpptulemusena päringu kiiremaks, sest arvuti ei pidanud viima läbi tekstiotsingut, vaid teadis, milliseid objekte pärida tuleb.

Lisas 1 on ära toodud osa veebirakenduse logi failist, mis annab selgust, kuidas töötab päringu vahemälu ilma teise taseme vahemälu kasutamata. Mälus hoitakse tulemused osaliselt, puuduvatele objektidele tehakse uued andmebaasi päringud. Mis tähendab andmebaasile suurt koormust.

```
<defaultCache
        maxElementsInMemory="20000"
        maxElementsOnDisk="0"
        eternal="false"
        timeToLiveSeconds="86400"
        overflowToDisk="false"
        memoryStoreEvictionPolicy="LRU"
        statistics="true"/>
```

Joonis 15. Vahemälu regiooni konfiguratsioon juhul, kui ruumi andmete jaoks on vähe.

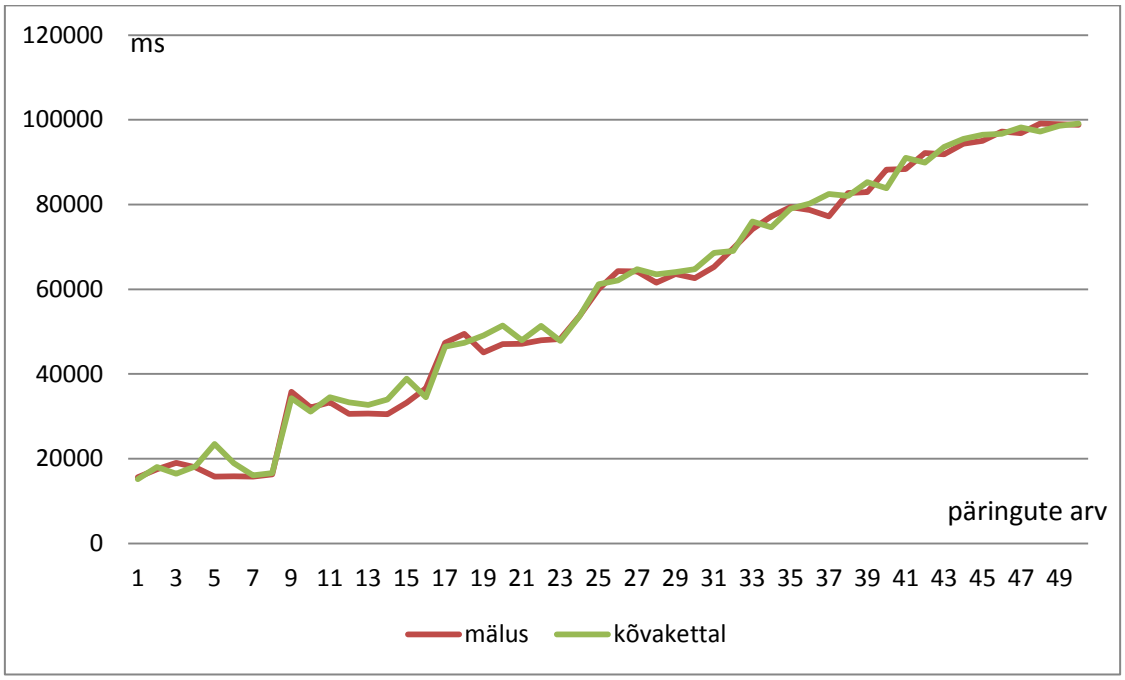
Katse 2:

Ehcache pakub võimalust salvestada andmeid kettaseadmele, juhul, kui lubatud mälu elementide arv ületatakse.

Joonisel 17 on näha vaikimisi mälupiirkonna konfiguratsioon, mille rakendumisel kõik andmed koheselt kettaseadmele salvestatakse.

Kui katse 1 puhul oleks märgitud overflowToDisk parameeter õigeks (true) ja parameeter maxElementsOnDisk oleks olnud suurem kui 0, oleks Ehcache kasutanud vahemälu lisamäluna kettaseadet.

Joonisel 16 on kujutatud graafik päringute ajalisest võrdlusest, kus kasutatakse arvuti mälu ja kettaseadet. Tulemustest on näha, et graafiku kõverad on peaaegu identsed. See tähendab, et ajalist kaotust ei teki, kui kasutada aeglasemat kettaseadme mälu arvuti mälu asemel.



Joonis 16. Vahemälu tulemuste salvestamise võrdlus salvestusseadme järgi.

```
<defaultCache
        maxElementsInMemory="0"
        maxElementsOnDisk="25000"
        eternal="false"
        timeToLiveSeconds="86400"
        overflowToDisk="true"
        memoryStoreEvictionPolicy="LRU"
        statistics="true"/>
```

Joonis 17. Ehcache kasutab kettaseadet andmete hoistamiseks.

5.6 Andmete muutmisega varundusstrateegia mõju päringu kiirusele

Peatükis 2.2.4 tutvustati vahemälu erinevaid varundusstrateegiaid. Selles peatükis uuritakse, kuidas käitub vahemälu andmete muutmisel ja kuidas see mõjutab päringu kiirust.

Katse:

Katse käigus pannakse vahemälu piirkond “movie“ kasutama strateegiat read/write.

```
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE, region =
"movie")
```

Seejärel laetakse vahemällu praegu andmebaasis olevad väärtused. Kasutatakse 100 esimest elementi andmetest. Päringu täitmise aeg mõõdetakse.

Pärast seda muudetakse kõik andmed eelpool päritud objektidel mis asuvad väljal “name”. Peale seda küsitakse sarnaselt algsele andmed ja selle täitmise aeg mõõdetakse. Tulemused on esitatud tabelis 4 ja veebirakenduse konsooli väljunditel.

Konsooli väljund andmete muutmisel:

DEBUG UpdateTimestampsCache:79 - Pre-invalidating space [Movie], timestamp: 5866732612042752
DEBUG EhcacheGeneralDataRegion:100 - key: Movie value: 5866732612042752
INFO sqltiming:357 - update Movie set name='Nimed on muudetud filmidel, katse 1' where movie_id<=100 {executed in 102 msec}
DEBUG UpdateTimestampsCache:99 - Invalidating space [Movie], timestamp: 5866732366991360
DEBUG EhcacheGeneralDataRegion:100 - key: Movie value: 5866732366991360

Konsooli väljund andmete pärimisel küsimisel:

DEBUG StandardQueryCache:131 - Checking cached query results in region: org.hibernate.cache.internal.StandardQueryCache
DEBUG EhcacheGeneralDataRegion:69 - key: sql: select movie0_.movie_id as movie_id1_4_, movie0_.date as date2_4_, movie0_.name as name3_4_, movie0_.rating_id as rating_i4_4_, movie0_.studio_id as studio_i5_4_ from Movie movie0_ where movie0_.movie_id<?; parameters: ; named parameters: {id=100}; transformer: org.hibernate.transform.CacheableResultTransformer@110f2
TRACE StandardQueryCache:215 - key.hashCode=327631398
TRACE StandardQueryCache:216 - querySpaces=[Movie]
DEBUG EhcacheGeneralDataRegion:69 - key: Movie
DEBUG UpdateTimestampsCache:128 - [Movie] last update timestamp: 5866732366991360, result set timestamp: 5866731993530368
DEBUG StandardQueryCache:143 - Cached query results were not up-to-date
INFO sqltiming:357 - select movie0_.movie_id as movie_id1_4_, movie0_.date as date2_4_, movie0_.name as name3_4_,
movie0_.rating_id as rating_i4_4_, movie0_.studio_id as studio_i5_4_ from Movie movie0_ where

movie0_.movie_id<=100 {executed in 2 msec}
DEBUG ConcurrentStatisticsImpl:411 - HHH000117: HQL: select m from Movie m where m.movieId<=:id, time: 42ms, rows: 100
DEBUG StandardQueryCache:104 - Caching query results in region: org.hibernate.cache.internal.StandardQueryCache; timestamp=5866732643995648

Päringu aeg enne andmete muutmist	65ms
Päringu aeg peale andmete muutmist	85ms

Tabel 6. Päringu ajad andmete muutmisel.

Konsooli väljundist saab välja lugeda, et Ehcache muudab movie piirkonna ajamärget peale andmebaasis olevate andmete muutmist.

Peale andmebaasis andmete muutmist tehakse päring 100-le esimesele filmile. Konsooli tulemustest võib välja lugeda, et Hibernate saab aru, et regiooni ajamärget on muudetud – see tähendab, et andmeid on vaja enne tagastamist vahemälus uuendada. Andmete uuendamiseks pöördutakse andmebaasi poole ja tehakse uus päring, regioonile lisatakse uus ajamärge.

Tabeli 4 tulemustest võib järeldada, et kui muudetakse andmeid läbi veebirakenduse, siis Hibernate ja Ehcache on sellest teadlikud ning andmeid muudetakse ka vahemälus. See omakorda tähendab lisapäringut ja seda, et andmete pärimine kestab pisut kauem.

5.7 Teise taseme vahemälu kasutamise puudused

5.7.1 Ebaühtlane tulemuste vananemine regioonide vahel.

Katse: Päringu vahemälu parameeter timeToLiveSeconds on määratud vananema hiljem kui teise taseme vahemälu regioonid, mida päringu vahemälu kasutab objektide saamiseks.

Tulemus: See tähendab, et kui vahemälu andmed on vananenud, siis need kustutatakse. See omakorda tähendab, et päringu saamisel teeb Hibernate identifikaatorite põhjal suure koguse päringuid andmebaasile. Juhtub sama olukord, mis eelnevalt on tehtud peatüki 5.5 katses 2, kui uuriti kuidas töötab vahemälu, kui andmed teise taseme vahemälust puuduvad.

5.7.2 Vahemälu andmete värskus

Hibernate peab järge andmete muudatuste kohta andmebaasis, mis on tehtud läbi veebirakenduse. Kui sama andmebaasi skeemi kasutab mõni teine rakendus või kui muudetakse andmeid otse andmebaasist, siis vahemälus olevaid tulemusi ei uuendata.

Katse: Pärime andmed ühe konkreetse objekti kohta läbi veebirakenduse. Seejärel muudame läbi PostgreSQL'i andmebaasi liidese vastava filmi nime ning pärast seda kuvame andmed uuesti läbi veebirakenduse.

Esimene Päring (HQL): SELECT m FROM Movie m WHERE m.movieId=1

Tulemusena tagastatakse film nimega "Avengers"

Teine päring andmebaasis (SQL): UPDATE movie SET name='Furious 7' WHERE movie_id = 1;

Andmebaasis vastavat rinda kuvades antakse tulemuseks film nimega "*Furious 7*".

Kolmas päring veebirakenduses (HQL): SELECT m FROM Movie m WHERE m.movieId=1

Tulemuseks vastab veebirakendus, et filmi nimi, mida päriti, oli "*Avengers*"

Nagu on näha tulemustest, siis otse andmebaasist muudetud andmeid vahemälus ei uuendata.

Sellise olukorra vältimiseks oleks soovitatav teha kõik muutmised läbi ühe veebirakenduse või muuta vahemälu elementide hoiustamise aega (timeToLiveSeconds) võimalikult lühikeseks.

6. Kokkuvõte

Töö eesmärgiks oli anda ülevaade Hibernate vahemälu õigest kasutamisest ja uurida vahemälu käitumist erinevate päringute ja koormuste rakendamisel Java veebirakendusele.

Töö käigus valmis õigesti konfigureeritud veebirakendus, mis kasutab edukalt vahemälu. Antud rakendust võib võtta näitena teiste rakenduste seadistamisel.

Töö tulemustest võib järeldada, et vahemälu kasutamine Java veebirakenduses muudab enamikul juhtudel selle reaktsiooniaega kiiremaks. Kõige suurema ajalise võidu andis vahemälu kasutamine andmete puhul, mida päritakse identifikaatorite järgi ning mida muudetakse harvadel juhtudel.

Vahemälu kasutamisega on võimalik kokku hoida andmebaasile saadetavate päringute pealt, saades korduvalt päritavad andmed otse mälust.

Vahemälu kasutamisel on andmete värskuse säilitamise seisukohast tähtis andmeid muuta läbi sama veebirakenduse – vahemälu ei reageeri andmemuudatusele, mida tehakse läbi teise rakenduse või otse andmebaasis.

Töö edasiarendusena võiks võrrelda teiste vahemälu teenusepakkujate teeke ja leida, milline teek sobib enim mõne rakenduse reaktsiooniaja optimeerimiseks. Katsete tegemiseks võiks kasutada spetsiaalset veebiserverit.

Töö käigus saavutati eesmärk muuta Java veebirakendus kiiremaks. Hibernate ja EhCache teekide kasutamine muutis rakenduse reaktsiooniaja paremaks. Samas saadi teada, milline oht valitseb andmete värskusele, kui kasutada vahemälu.

Summary

The purpose of the thesis was to give an overview on the correct usage of Hibernate cache and to study the behavior of Java web application when different queries are run with or without high user traffic.

The outcome of the thesis was correctly configured web application which successfully uses the cache. This application can be used as an example for setting up other applications.

In conclusion, for the most cases Java web application's response time can be done faster when the cache is used. The biggest time was saved when the cache was used with the data which was returned using unique identifiers and when that data was not updated frequently.

Also, repeated database queries can be reduced, when the cache is used.

Finally, it is essential to interact with the data through the same application – the cache does not respond to what is being changed through another application.

Kasutatud kirjandus

[1] Vahemälu (cache) definitsioon. [Online] http://www.vallaste.ee/index.htm?Type=UserId&otsing=551 (28.04.2015)

[2] Oliver White. Java Tools and Technologies Landscape for 2014. Object-Relational Mapping (ORM) frameworks. [Online] http://zeroturnaround.com/rebellabs/java-tools-and-technologies-landscape-for-2014/10/ (21.05.2014)

[3] Prof. Kuldar Taveter. Tarkvarasüsteemi nõuete inseneeria, 22 [Online] http://maurus.ttu.ee/sts/wp-content/uploads/2014/09/IDK0071-Loeng-2-Tarkvaras%C3%BCsteemi-n%C3%B5uete-inseneeria.pdf (23.05.15)

[4] Hibernate tutorial. Hibernate – caching [Online] http://www.tutorialspoint.com/hibernate/hibernate_caching.htm (29.04.2015)

[5] Krishna Srinivasan. What is Hibernate Caching? First-level cache [Online] http://www.javabeat.net/introduction-to-hibernate-caching/ (07.10.2007)

[6] Cache-provider/concurrency-strategy compatibility [Online] https://docs.jboss.org/hibernate/orm/3.3/reference/en/html/performance.html#performance-cache-compat-matrix (29.04.2015)

[7] Hibernate: Second Level Cache – EhCache [Online] https://anirbanchowdhury.wordpress.com/2012/07/23/hibernate-second-level-cache-ehcache/ (21.05.15)

[8] Apache JMeter 2.13 [Online] http://jmeter.apache.org/

[9] Hibernate JConsole Plugin (hibernate-jconsole) - 1.0.7 [Online] http://hibernate-jcons.sourceforge.net/

[10] Alex Miller. Hibernate query cache considered harmful? [Online] http://tech.puredanger.com/2009/07/10/hibernate-query-cache/ (10.07.2009)

Lisa 1 – Väljavõte konsooli väljundist kui vahemälu võimalusi kasutatakse osaliselt

2015-05-19 00:47:21 TRACE StandardQueryCache:266 - tuple is Object[1]; returnTypes is Type[1]

2015-05-19 00:47:21 INFO sqltiming:357 - select person0_.person_id as person_i1_5_1_, person0_.born as born2_5_1_, person0_.firstname

as firstnam3_5_1_, person0_.gender_id as gender_i5_5_1_, person0_.lastname as lastname4_5_1_,

gender1_.gender_id as gender_i1_2_0_, gender1_.name as name2_2_0_ from Person person0_ left

outer join Gender gender1_ on person0_.gender_id=gender1_.gender_id where person0_.person_id=21984

{executed in 1 msec}

2015-05-19 00:47:21 TRACE StandardQueryCache:266 - tuple is Object[1]; returnTypes is Type[1]

2015-05-19 00:47:21 INFO sqltiming:357 - select person0_.person_id as person_i1_5_1_, person0_.born as born2_5_1_, person0_.firstname

as firstnam3_5_1_, person0_.gender_id as gender_i5_5_1_, person0_.lastname as lastname4_5_1_,

gender1_.gender_id as gender_i1_2_0_, gender1_.name as name2_2_0_ from Person person0_ left

outer join Gender gender1_ on person0_.gender_id=gender1_.gender_id where person0_.person_id=21985

{executed in 1 msec}

2015-05-19 00:47:21 TRACE StandardQueryCache:266 - tuple is Object[1]; returnTypes is Type[1]

2015-05-19 00:47:21 INFO sqltiming:357 - select person0_.person_id as person_i1_5_1_, person0_.born as born2_5_1_, person0_.firstname

as firstnam3_5_1_, person0_.gender_id as gender_i5_5_1_, person0_.lastname as lastname4_5_1_,

gender1_.gender_id as gender_i1_2_0_, gender1_.name as name2_2_0_ from Person person0_ left

outer join Gender gender1_ on person0_.gender_id=gender1_.gender_id where person0_.person_id=21986

{executed in 1 msec}

2015-05-19 00:47:21 TRACE StandardQueryCache:266 - tuple is Object[1]; returnTypes is Type[1]

2015-05-19 00:47:21 ERROR HelloController:74 - Totaltime of person QUERY 29906