Tampereen ammattikorkeakoulu

Tehokas tietokantayhteys monimutkaisissa Java-sovelluksissa

Janne Leppäharju

OPINNÄYTETYÖ
Huhtikuu 2019

Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

LEPPÄHARJU, JANNE:
Tehokas tietokantayhteys monimutkaisissa Java-sovelluksissa

Opinnäytetyö 27 sivua, joista liitteitä 3 sivua
Huhtikuu 2019

Opinnäytetyön tarkoituksena oli tutustua monimutkaisen Java-sovelluksen tietokantayhteyden optimoimiseen ja tavoitteena oli kehittää esimerkkisovellus, jossa hyödynnetään hyväksi havaittuja optimointi keinoja.

Tutkimuksen perusteella kehitettiin video vuokraus sovellus, joka hyödynsi keskeisiä monimutkaisten Java-sovelluksien käyttämiä teknologioita. Sovellusta hyödynnettiin opinnäytetyössä tehokkaiden tietokantaratkaisujen todentamiseen ja testaamiseen. Testeistä kävi ilmi, että jo pienellä työkalujen optimoimisella saadaan suuret hyödyt, kun tietokannassa on paljon dataa.

Opinnäytetyön tuloksia ja suosituksia voidaan hyödyntää uusien monimutkaisten Java-sovelluksien tietokanta hallintajärjestelmien ja työkalujen valinnassa sekä uusien ja vanhojen sovellusten suorituskyvyn optimoimiseen.

Asiasanat: java, hibernate, relaatiotietokanta

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information and Communication Technology
Software Engineering

LEPPÄHARJU, JANNE:
Efficient database connectivity for complex Java applications

Bachelor's thesis 27 pages, appendices 3 pages
April 2019

The purpose of the thesis was to get to know the database connection optimization of complex Java-based application and to develop an example application that utilizes the proven optimization methods.

Based on the research, a video rental application was developed that utilized the key technologies used by complex Java applications. The application was used in the thesis to verify and test efficient database solutions. The tests showed that only minor optimization makes huge different when there are lots of data in the database.

The results and recommendations of the thesis can be utilized in selecting new complex Java application database management systems and tools and optimizing the performance of new and old applications.

Key words: java, hibernate, relational database

SISÄLLYS

LYHENTEET JA TERMIT

DBMS	Tietokannan hallintajärjestelmä, Database management system
RDBMS	Relaatiotietokannan hallintajärjestelmä, Relational database management system
API	Ohjelmointirajapinta, Application programming interface
SQL	Jäsennelty kyselykieli, Structured Query Language
JDBC	Javan tietokanta rajapinta, Java Database Connectivity
ORM	Olio relaatio kartoitus, Object Relational Mapping
JPA	Javan persistence rajapinta, Java Persistence API
POJO	Tavallinen Java-luokka, jonka toteutukseen ei liity infrastruktuurin asettamia rajoitteita tai vaatimuksia, Plain Old Java Object
DTO	Olio, joka kuljettaa tietoa luokkien välillä, Data transfer object

1 JOHDANTO

Monimutkaisten Java-sovelluksien suorituskykyongelmat johtuvat usein huonosti toteutetuista tietokantayhteyksistä. Suorituskykyä voidaan parantaa optimoimalla ja käyttämällä valmiita työkaluja yhteyksien muodostamiseen, hallinnointiin ja tiedonsiirtoon.

Opinnäytetyössä tutustutaan tietokantoihin, monimutkaisen Java-sovelluksen tietokantayhteyden optimoimiseen ja työkaluihin sen ympärillä. Tutustuminen tapahtuu kehitetyn videovuokraus esimerkkisovelluksen kautta, koska toimeksiantajan järjestelmää ei voida luottamuksellisuussyistä käyttää esimerkkinä.

Opinnäytetyöraportin alkuvaiheessa tutustutaan teknologioihin monimutkaisen Java-sovelluksen ympärillä, keskivaiheessa annetaan vinkit työkalujen optimoimiseen ja lopussa käydään läpi, miten opinnäytetyön sujui.

2 JAVA OHJELMISTOALUSTANA

Java on tehokas, yleiskäyttöinen ohjelmointiympäristö ja teknologiaperhe. Se on yksi yleisimmin käytetyistä ohjelmointikielistä maailmassa ja se on ollut erityisen menestyksekäs liike- ja yrityskäytössä. (Flanagan & Evans 2014, 1.)

Kappaleessa käydään läpi Java-teknologiaperheen ja siihen liittyvien kirjastojen ominaisuuksia. Kappale on jaettu kolmeen aliotsikkoon.

2.1 Java-ohjelmointikieli

Sun Microsystemsillä Net-konkareiden James Goslingin ja Bill Joyn ohjauksessa kehitetty Java-ohjelmointikieli suunniteltiin laitteistoriippumattomaksi ohjelmointikieleksi, joka on samalla riittävän turvallinen sekä tarpeeksi tehokas korvaamaan natiivisti suoritettavan koodin. (Niemeyer & Leuck 2013, 2.)

Java-ohjelmointikieli on ihmisluettava, luokkapohjainen ja oliosuuntautunut. Se on helppolukuista, helposti kirjoitettavaa ja perustuu teollisuuden kokemuksiin kielistä kuten C++, siten, että aikaisempien ohjelmointikielien monimutkaiset ominaisuudet on pyritty poistamaan samalla kuin toimivat ominaisuudet on jätetty. (Flanagan & Evans 2014, 4.)

Ohjelmointikielenä Javalla on suhteellisen konservatiivinen muotoilu ja se muuttuu hitaasti. Nämä ominaisuudet on määritelty suojelemaan investointeja, joita yritykset ovat tehneet Java-teknologialle. (Flanagan & Evans 2014, 4.)

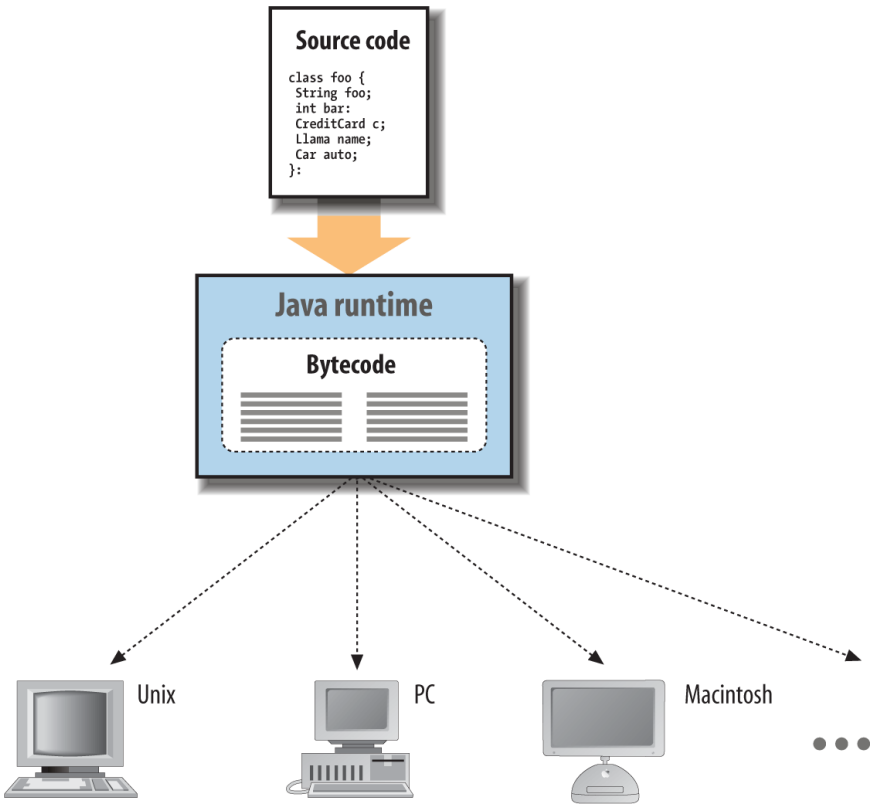
Vuonna 1996 julkaistun ensimmäisen version jälkeen se on saanut useita uusia versioita, mutta sitä ei ole missään vaiheessa kirjoitettu kokonaan uudelleen, joten ensimmäiseen versioon tehdyt suunnitteluratkaisut vaikuttavat edelleen. (Flanagan & Evans 2014, 4.)

Java on sekä käännettävä että tulkittava ohjelmointikieli. Toisin kun C tai C++ Java-lähdekoodia ei käännetä suoraan konekielelle vaan se käännetään ensin

tavukoodiksi, jonka Javan ajonaikainen tulkki suorittaa. (Niemeyer & Leuck 2013, 4.)

Ajonaikainen järjestelmä suorittaa kaikki normaalit prosessorin aktiviteetit turvallisessa virtuaalisessa ympäristössä. Se suorittaa pinopohjaiset käskysarjat ja hallitsee muistia kuten käyttöjärjestelmä. Se luo ja manipuloi primitiivisiä tietotyyppejä ja lataa ja kutsuu hiljattain viitattuja koodilohkoja. Toiminnot suoritetaan tiukasti määritellyn avoimen määrittelyn mukaisesti, jonka voi toteuttaa jokainen, joka haluaa luoda Java-yhteensopivan virtuaalikoneen. Virtuaalikoneen ja kielen määritelmät tarjoavat yhdessä täydellisen eritelmän. Ei ole olemassa Java-kielen perusominaisuuksia, jotka olisivat määrittämättömiä tai toteutuksesta riippuvaisia. Esimerkiksi Java määrittää kaikkien sen primitiivisten tietotyyppien koot ja matemaattiset ominaisuudet, eikä jätä niitä alustan toteutettavaksi. (Niemeyer & Leuck 2013, 4 - 5.)

Java-tulkki on suhteellisen kevyt ja se voidaan toteuttaa alustalle sopivassa muodossa. Tulkkia voidaan ajaa erillisessä sovelluksessa tai se voidaan upottaa toiseen ohjelmistoon, kuten verkkoselaimeen. Samaa Java-sovelluksen tavukoodia voidaan ajaa millä tahansa alustalla, joka tarjoaa Java-ajoympäristön, kuten kuvassa 1 näkyy. Tämä mahdollistaa sen, ettei sovelluksesta tarvitse tehdä eri versioita jokaiselle alustalle. (Niemeyer & Leuck 2013, 5.)



KUVA 1. Java ajoympäristö (Niemeyer & Leuck 2013, 5)

2.2 Java-ekosysteemi

Java-ohjelmointikieli on helposti opittava ja sisältää suhteellisen vähän erikoisuuksia verrattuna muihin kieliin. JVM eli Java virtuaalikone tarjoaa vakaan, alustariippumattoman ja tehokkaan pohjan Java koodin suorittamiseen. Nämä kaksi yhdistettyä teknologiaa tarjoavat vahvan perustan, johon yritykset voivat kohdistaa oman kehitystyönsä. Lisäksi Javan ympärille on kasvanut erittäin suuri kolmannen osapuolen kirjastojen ja komponenttien ekosysteemi, joka tarkoittaa sitä, että kehitystiimit voivat hyötyä suuresti olemassa olevista rajapinnoista ja ajureista lähes kaikille keksityille teknologioille. Esimerkiksi kaikille perinteisille relaatiotietokannoille ja NoSQL:lle löytyy omat Java rajapinnat. Nämä seikat ovat olleet merkittävänä tekijänä Java-teknologioiden käyttöönotossa yrityksissä. (Flanagan & Evans 2014, 7.)

2.3 JavaFX-visualisointityökalu

JavaFX on joukko grafiikka ja media paketteja, jotka mahdollistavat kehittäjien suunnitella, luoda, testata, korjata ja tuottaa sovelluksia, jotka toimivat samalla tavalla eri alustoilla. (Oracle, 2019.)

JavaFX-kirjasto on kirjoitettu Java rajapinnaksi, joten JavaFX-sovelluskoodi voi viitata mihin tahansa rajapintaan, mistä tahansa Java kirjastosta. Esimerkiksi JavaFX-sovellukset voivat käyttää Java rajapinta kirjastoja päästäkseen järjestelmäkapasiteetteihin ja muodostaakseen yhteyden palvelinpohjaisiin sovelluksiin. (Oracle, 2019.)

JavaFX-sovellusten ulkoasu voidaan toteuttaa, joko suoraan ohjelmakoodissa Java-kielellä tai erottamalla logiikka ja käyttöliittymä käyttämällä käyttöliittymän ohjelmointiin FXML-merkintäkieltä. FXML on XML-pohjainen merkintäkieli Java-objektikaavioiden rakentamiseen (Oracle, 2012).

3 TIETOKANNAT YLEISELLÄ TASOLLA

Jokaisella organisaatiolla on tietoja, jotka sen on säilytettävä ja hallittava täyttääkseen vaatimukset. Yrityksen on esimerkiksi kerättävä ja ylläpidettävä henkilöstöön liittyviä tietoja. Tietojen pitää olla saatavilla niitä tarvitseville. Tarvitaan siis tietojärjestelmä ja useimmat yritykset käyttävät tähän tarkoitukseen tietokantoja.

Tietokanta on jäsennelty kokoelma tietoa. Se voi olla mitä vain yksinkertaisesta ostoslistasta kuvagalleriaan tai suuria määriä tietoa yritysverkossa. Tietokantaan tallennettujen tietojen lisäämiseen, hakuun ja käsittelyyn tarvitaan tietokannan hallintajärjestelmä, kuten esimerkiksi MySQL tai Oracle. (What is MySQL, 2019.)

Relaatiotietokanta tallentaa tiedot erillisiin tauluihin sen sijaan, että kaikki tiedot sijoitettaisiin yhteen suureen varastoon. Tietokantarakenteet ovat järjestetty fyysisiin tiedostoihin, jotka ovat optimoitu nopeiksi. Tietokenttien välisille suhteille asetetaan säännöt, kuten yhden suhde yhteen, yhden suhde moneen tai monen suhde moneen. Tietokanta noudattaa määritettyjä sääntöjä siten, että tiedot pysyvät eheinä sovellusta käytettäessä. (What is MySQL, 2019.)

Kappaleessa käydään läpi yleisimmät tietokanta hallintajärjestelmät. Ne ovat jaettu kahteen kategoriaan, SQL- ja NoSQL-tietokantoihin.

3.1 SQL-tietokannat

SQL-tietokannat ovat tietokantoja, jotka käyttävät SQL-kieltä tietojen hallinnointiin. SQL on yleisin standardoidu kieli tietokantojen hallinnointiin. SQL on määritelty ANSI/ISO SQL standardin mukaisesti, jota on kehitetty jo vuodesta 1986. (What is MySQL, 2019.)

3.1.1 MySql-tietokanta hallintajärjestelmä

MySQL on suosituin avoimen lähdekoodin SQL-tietokanta hallintajärjestelmä. Sitä kehittää, jakaa ja tukee Oracle. MySQL-tietokannat ovat relaatiotietokantoja ja ne ovat suunniteltu ennen kaikkea käsittelemään suuria määriä tietoa nopeasti. (What is MySQL, 2019)

3.1.2 Oracle-tietokanta hallintajärjestelmä

Oracle on relaatiotietokanta hallintajärjestelmä, joka on laajentanut relaatiomallia olio-relaatiomalliksi, jotta sen olisi mahdollista tallentaa monimutkaisia liiketoimintamalleja relaatiotietokantaan. Se toteuttaa olio-orientoituja ominaisuuksia, kuten käyttäjän määrittämät tyypit, periytymisen ja polymorfismin. (Introduction to Oracle Database, 2015.)

3.1.3 Postgre-tietokanta hallintajärjestelmä

PostgreSQL on tehokas, avoimen lähdekoodin olio-relaatiotietokanta hallintajärjestelmä, joka käyttää ja laajentaa SQL-kieltä yhdistettynä moniin ominaisuuksiin, jotka tallentavat ja skaalauttavat turvallisesti kaikkein monimutkaisimmat tietokuormat. PostgreSQL:n perusta kehitettiin vuonna 1986 osana POSTGRES-hanketta Kalifornian yliopistossa Berkeleyssä ja sen kehitys jatkuu edelleen. (What is PostgreSQL, 2019.)

PostgreSQL on saavuttanut suuren suosionsa toimivasta arkkitehtuurista, luotettavuudesta, tietojen eheydestä, laajoista ominaisuuksista, laajennettavuudesta ja omistautuneesta kehittäjäyhteisöstä ohjelmistojen takana. PostgreSQL toimii kaikilla suurimmilla käyttöjärjestelmillä, se on ollut ACID-yhteensopiva vuodesta 2001 lähtien ja sillä on tehokkaita lisäosia. (What is PostgreSQL, 2019.)

3.2 NoSQL-tietokannat

NoSQL-käsitteellä kuvataan tietokantoja, jotka poikkeavat tutusta SQL-relaatiomallista. NoSQL-tietokanta rakennetaan tiettyä tietomallia kohden ja niissä on joustavia skeemoja nykyaikaisten sovellusten rakentamiseen. NoSQL-tietokannat tunnetaan laajasti niiden helppokäyttöisyyden, toimivuuden ja suorituskyvyn ansiosta. NoSQL-tietokannat käyttävät erilaisia tietomalleja, kuten asiakirja, kaavio, avainarvo, muisti ja haku. (What is NoSQL, 2019.)

3.2.1 MongoDB-tietokanta hallintajärjestelmä

MongoDB on asiakirjatietokanta, joka on skaalautuva ja joustava, sekä tarjoaa tehokkaan haun ja indeksoinnin. MongoDB tallentaa tiedot JSON-tyylisille dokumenteille, joka tarkoittaa, että kentät voivat vaihdella eri dokumenteilla ja tietorakennetta voidaan muuttaa haluttaessa. (What is MongoDB, 2019.)

4 PYSYVYYS OLIO-RELAATIO MENETELMÄLLÄ

4.1 Pysyvyys

Lähes kaikki sovellukset vaativat pysyviä tietoja. Pysyvyys on yksi sovelluskehityksen perusperiaatteista. Jos tietojärjestelmä hävittäisi kaiken tiedon sammutuksen yhteydessä, niin se olisi hyvin epäkäytännöllinen. Olion pysyvyys tarkoittaa, että yksittäiset objektit voivat elää kauemmin kuin sovellusprosessi; ne voidaan tallentaa tietovarastoon ja luoda uudelleen myöhemmin. Java-sovelluksen pysyvyydestä puhuttaessa, tarkoitetaan yleensä olio esiintymien kartoittamista ja tallentamista SQL-tietokantaan. Pysyvyys käsite tunnetaan arkikielessä myös nimellä persistenssi. (Bauer, King, Gregory 2013, 4.)

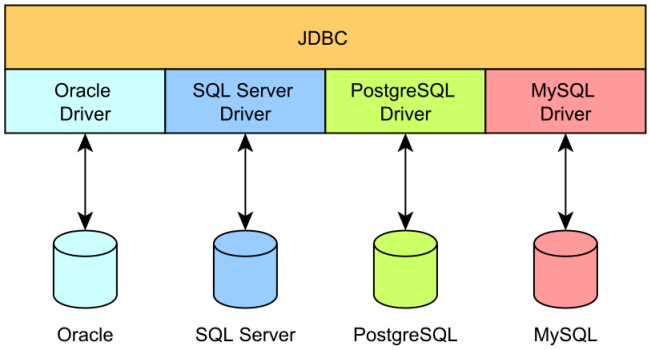
4.2 Olio-relaatio kartoitus

Olio-relaatio kartoitus toteuttaa automaattisesti pysyvyyden Java-sovelluksen olioille SQL-tietokannan tauluihin, käyttäen metatietoja, jotka kuvailevat yhteyden Java-sovellus luokkien ja tietokantamallin välillä. Toisin sanoen se toimii muuntimena eri esitysmuotojen välillä. (Bauer, King, Gregory 2013, 16.)

5 TIETOKANTAYHTEYS TEKNOLOGIAT JAVA-SOVELLUKSISSA

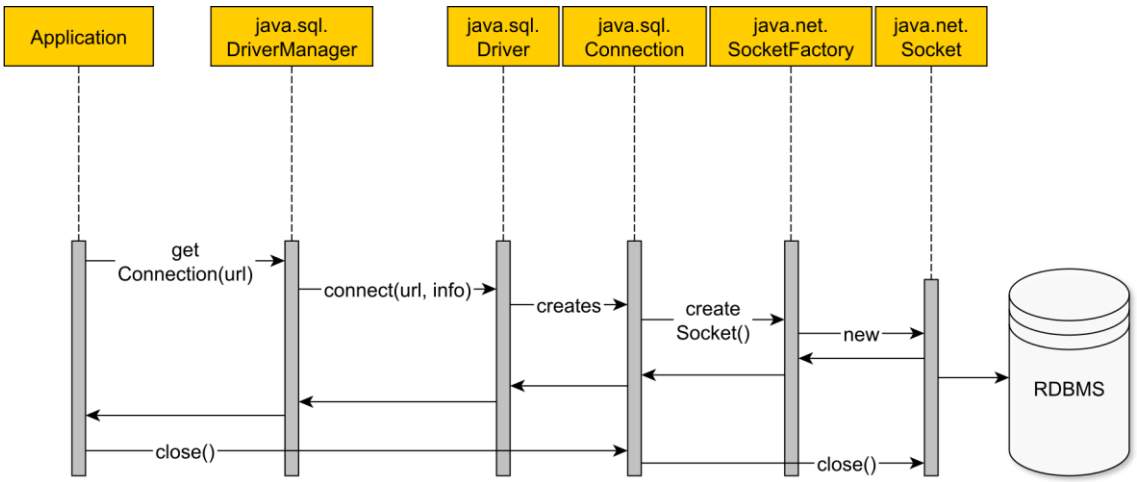
5.1 Tietokantayhteys JDBC:llä

JDBC (Java Database Connectivity) on ohjelmointirajapinta (API), joka tarjoaa yleisen rajapinnan tietokantapalvelimen kanssa kommunikointiin. Kaikki verkkologiikka ja tietokannasta riippuvat kommunikointi protokollat on piilotettu tietokantavalmistaja kohtaisten JDBC ohjelmistorajapintojen taakse. Tästä syystä kaikki JDBC-rajapinnat on toteutettava tietokantavalmistajan erityisvaatimusten mukaisesti. Kuvassa 2 on kuvattu JDBC API:n ja tietokantavalmistaja kohtaisten JDBC rajapintojen välinen arkkitehtuuri. (Mihalcea 2016, 12.)



KUVA 2. JDBC liitäntä arkkitehtuuri (Mihalcea 2016, 12)

Java.sql.Driver luokka on pääpiste, sovelluksen ollessa vuorovaikutuksessa JDBC API:n kanssa. Vaikka Driver on todellinen tietokantayhteyden tarjoaja, on kätevämpää käyttää java.sql.DriverManageria, koska se voi myös löytää oikean JDBC-ajurin tietokantayhden URL-osoitteen perusteella. Kommunikoinnin aloittamiseksi Java-sovelluksen on ensin saatava java.sql.Connection. Kuten kuvasta 3 voidaan huomata, kysyttäessä yhteyttä DriverManagerilta, DriverManager pyytää uutta fyysistä yhteyttä alla olevalta Driverilta. (Mihalcea 2016, 12-13)



KUVA 3. JDBC-yhteys (Mihalcea 2016, 13)

Java-sovellus koodissa tietokantayhteyden luonti ja asiakastaulusta asiakkaiden etu- ja sukunimien kysyminen tapahtuu JDBC:tä käyttäen kuvan 4 mukaisesti. Alussa pyydetään yhteyttä DriverManager luokalta, antaen sille tarvittava URL-osoite tietokantaan, tietokannan käyttäjänimi ja salasana. Kun yhteys on luotu, voidaan SQL-kieltä käyttäen kysyä haluttuja tietoja tietokannasta.

```
25          String sql = "SELECT * FROM customer";
26
27          try {
28              Class.forName("com.mysql.cj.jdbc.Driver");
29
30              connection = DriverManager.getConnection(dbUrl, user, password);
31
32              statement = connection.createStatement();
33
34              ResultSet resultSet = statement.executeQuery(sql);
35
36              while(resultSet.next()) {
37                  System.out.println(resultSet.getString("first_name") + " " + resultSet.getString("last_name"));
38              }
39              resultSet.close();
40
41          } catch (SQLException | ClassNotFoundException e) {
42              e.printStackTrace();
43          } finally {
44              try {
45                  connection.close();
46              } catch (SQLException e) {
47                  e.printStackTrace();
48              }
49          }
```

KUVA 4. Asiakastietojen haku käyttäen JDBC:tä

5.2 JPA ja Hibernate tietokantayhteyden muodostamisessa

JPA (Java Persistence API) on määritelmä. Se kuvaa rajapinnat, joita asiakaspään sovellukset käyttävät ja standardin olio-relaatiokartoitus metatiedon. JPA myös määrittää, kuinka nämä määritykset JPA-tarjoajien pitäisi toteuttaa. (Mihalcea 2016, 120.)

Olio-relaatiokartoitukseen vaadittava metatieto saadaan annotoimalla persistoitava luokka kuvan 5 osoittamalla tavalla. Entity annotaatio osoittaa, että luokan instanssi tallennetaan tietokantaan. Table annotaatio yhdistää luokan tietokanta tauluun. Column annotaatio yhdistää luokka muuttujan tietokantataulun sarakkeeseen. Id ja GeneratedValue annotaatiot kertovat, että id muuttuja on luokan avainarvo ja lisättäessä asiakkaita tauluun, id muuttujan arvo kasvaa automaattisesti. OneToOne annotaatio osoittaa, että asiakkaan osoite on erillisessä taulussa, sekä asiakkaalla on yksi osoite. OneToMany annotaatio osoittaa, että asiakkaan vuokraukset, ovat erillisessä taulussa, sekä niitä voi olla useita yhdellä asiakkaalla.

```
28 @Entity
29 @Table(name="customer")
30 public class Customer {
31
32     @Id
33     @GeneratedValue(strategy=GenerationType.IDENTITY)
34     @Column(name="id")
35     private int id;
36
37     @Column(name="first_name")
38     private String firstName;
39
40     @Column(name="last_name")
41     private String lastName;
42
43     @Column(name="email")
44     private String email;
45
46     @Column(name="date_of_birth")
47     @Temporal(TemporalType.DATE)
48     private Date dateOfBirth;
49
50     @OneToOne(cascade=CascadeType.ALL)
51     @JoinColumn(name="customer_address_id")
52     private Address address;
53
54     @OneToMany(fetch=FetchType.LAZY, mappedBy="customer", cascade=CascadeType.ALL)
55     private List<Rental> rentals;
```

KUVA 5. JPA annotoitu POJO

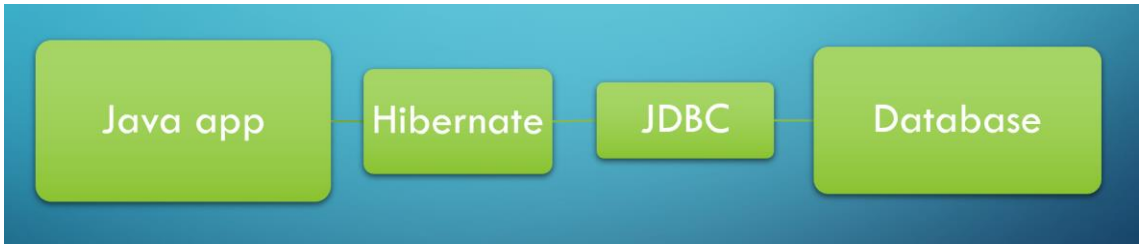
Hibernate on kunnianhimoinen hanke, jonka tavoitteena on tarjota täydellinen ratkaisu pysyvien tietojen hallintaan Java-sovelluksissa. Hibernate ei nykyään ole pelkästään ORM-palvelu, vaan myös kokoelma hallintatyökaluja, jotka ulottuvat ORM:n ulkopuolelle. (Bauer, King, Gregory 2013, 19.)

Hibernate ORM koostuu ytimestä, joka on peruspalvelu pysyvyydelle SQL-tietokantojen ja alkuperäisen patentoidun API:n kanssa. Hibernate ORM on vanhimpana projektina perustana muille projekteille. Hibernate ORM toimii kaikilla Java EE sovelluspalvelimilla, Swing ja JavaFx-sovelluksissa, yksinkertaisessa servlet-säiliössä ja niin edelleen. (Bauer, King, Gregory 2013, 19-20.)

Hibernate EntityManager on Hibernaten toteutus standardille Java Persistence API:lle. Se on valinnainen moduuli Hibernate ORM:n päälle. (Bauer, King, Gregory 2013, 20.)

Hibernate OGM on uusin Hibernate projekti, joka on olio/ruudukko kartoittaja. Se tarjoaa JPA tuen NoSQL ratkaisuille, käyttäen vanhaa Hibernaten ydintä, mutta persistoi kartoitetut kokonaisuudet avain/arvo, dokumentti tai kaavio tietokantoihin. (Bauer, King, Gregory 2013, 20.)

Hibernate on uusi taso tietokantayhteyden muodostamiseen ja se käyttää aiemmin esille tullutta JDBC:tä tietojen hakemiseen tietokannasta. Kuvassa 6 on esitetty yhteys Java-sovellukselta tietokantaan käyttäen Hibernatea.



KUVA 6. Hibernate yhteys

Hibernate yksinkertaistaa ja vähentää koodia, joka huomataan hyvin, kun verrataan 16 asiakastietojen hakua JDBC:llä, kuvassa 7 esiintyvään Hibernate hakuun.

```
28          Session session = sessionFactory.getCurrentSession();
29
30          try {
31              session.beginTransaction();
32              Query<Customer> query = session.createQuery("from Customer");
33
34              List<Customer> customer = query.getResultList();
35
36              session.getTransaction().commit();
37
38              session.close();
39          }
```

KUVA 7. Asiakastietojen haku käyttäen Hibernatea.

6 HIBERNATEN OPTIMOINTI

Kappaleessa käydään läpi muutama yleinen keino, joilla voi tehostaa Hibernaten käyttöä Java-sovelluksessa. Jo pienellä Hibernaten optimoimisella voi olla suuri vaikutus Java-sovelluksen suorituskykyyn, kun tietokannassa on tuhansia rivejä tietoa.

6.1 Tietokannasta vain tarvitut tiedot

Ylimääräisen tiedon hakeminen tietokannasta on suurin suorituskykyä alentava ongelma JPA:n ja Hibernaten käytössä. Ongelma syntyy, koska JPA mahdollistaa sen. ManyToOne ja OneToOne suhteet haetaan oletusarvoisesti käyttäen Eager-hakua, eli esimerkiksi haettaessa asiakkaan nimeä, niin kuvassa 8 esiintyvä asiakkaan osoite haetaan myös omasta taulustaan. Tästä voi syntyä ongelmia, jos osoite luokassa on lisäksi suhteita muihin tauluihin. Suhde ketjusta voi syntyä pitkä ja valtava määrä tietoa haetaan tietokannasta. (Mihalcea, 2019.)

```
50    @OneToOne(cascade=CascadeType.ALL)
51    @JoinColumn(name="customer_address_id")
52    private Address address;
```

KUVA 8. Eager-haettava OneToOne suhde

Ongelman voi korjata helposti lisäämällä JPA-annotaatioon ”fetch=FetchType.LAZY”, joka määrittää, että osoite tiedot haetaan vain erikseen pyydettäessä. Kuvassa 9 on esitettynä korjattu annotaatio.

```
50    @OneToOne(fetch=FetchType.LAZY, cascade=CascadeType.ALL)
51    @JoinColumn(name="customer_address_id")
52    private Address address;
```

KUVA 9. Lazy-haettava OneToOne suhde

Toinen tärkeä huomio on, että kun haetaan tietoja JPA:lla ja Hibernatella, niin on tärkeää erottaa käyttötapaukset, jotka tarvitsevat kokonaisuuksia eli entityjä, verrattuna niihin, jotka pärjäävät DTO-projektioilla. Nyrkkisääntönä voidaan pitää sitä, että jos halutaan tallentaa, päivittää tai poistaa tietueita, on entiteettien haku

kätevää. Jos kuitenkin halutaan vain esittää tietoja, eikä niitä tarvitse jatkossa muokata, niin DTO-projektioinnin käyttö on suotavempaa. Toisin, kun entiteettien haku, DTO-projektio mahdollistaa sarakkeiden määrän määrittämisen, jotka haetaan tietokannasta, joka taas voi nopeuttaa kyselyjä merkittävästi. (Mihalcea, 2019.)

Lisäksi kannattaa rajoittaa haettavien tietueiden määrää. Jos tiedot on tarkoitus näyttää käyttöliittymässä, on jo olemassa raja, kuinka paljon tietoja voidaan näyttää yhdellä näkymällä, joten kaikki muu ylimääräinen vaikuttaa sovelluksen suorituskykyyn. Data pyrkii kasvamaan ajan myötä ja jos haun tulosjoukkoa ei rajoiteta, haettavan datan määrä kasvaa suuremmaksi ja suuremmaksi. Jos halutaan ennakoitavissa olevia vasteaikoja, kyselyn tulosarjojen rajoittaminen on hyvä tapa toimia. (Mihalcea, 2019.)

6.2 Kyselyn haun koon optimointi

Hibernate käy läpi JDBC:n ResultSet:n ja rakentaa sen avulla listan entiteeteistä tai DTO olioista. FetchSize määrittää edestakaisten matkojen määrän, joita tarvitaan koko ResultSet:n noutamiseen.

Jos käytössä on PostgreSQL tai MySQL, niin ResultSet haetaan yhdellä tietokanta kierroksella, joten oletus haun koko on yleensä paras vaihtoehto, mutta jos käytössä on Oracle, niin oletus haun koko on vain 10 tietuetta. Joten haettaessa 100 tietuetta, joutuu Hibernate tekemään 10 kierrosta saadakseen haettua koko ResultSet:n. (Mihalcea 2016, 280.)

Kuvassa 10 on esimerkki, miten haun koko määritellään kyselyn yhteydessä. Kyselyyn lisätään QueryHints.HINT_FETCH_SIZE määritys ja haluttu haun koko.

```
List<PostCommentSummary> summaries = entityManager.createQuery(
    "select new " +
    "   com.vladmihalcea.book.hpjp.hibernate.fetching.PostCommentSummary( " +
    "       p.id, p.title, c.review ) " +
    "from PostComment c " +
    "join c.post p")
.setFirstResult(pageStart)
.setMaxResults(pageSize)
.setHint(QueryHints.HINT_FETCH_SIZE, pageSize)
.getResultList();
```

KUVA 10. Haun koon määritys

7 POHDINTA

Opinnäytetyön tarkoituksena oli tutustua monimutkaisen Java-sovelluksen tietokantayhteyden optimoimiseen ja tavoitteena oli kehittää esimerkkisovellus, jossa hyödynnetään hyväksi havaittuja optimointi keinoja.

Tietokantayhteyden tehokkuus on avain asemassa Java-sovelluksen hyvän käyttökokemuksen saavuttamisessa. Käyttäjät ovat tottuneet alle 200 millisekunnin vaste aikoihin ja jos tietokanta kysely kestää huomattavan pitkään, varsinkin, jos käyttäjälle ei anneta sopivaa ilmoitusta latauksesta, turhautuu käyttäjä nopeasti. Tästä syystä tietokantayhteyksiä halutaan optimoida ja miksi aihe valittiin opinnäytetyöksi.

Opinnäytetyön esimerkkisovellus osuus eteni hyvin ja sitä oli mielekästä tehdä, mutta kirjoitus vaihe eteni harmillisen hitaasti. Esimerkkisovellus oli videovuokraus sovellus, jonka UI toteutettiin JavaFx:llä ja siinä käytettiin kaikkia teknologioita, joita opinnäytetyössä käsiteltiin. Tämän lisäksi tutustuin sovellusta varten Maven projektinhallinta työkaluun ja Spring riippuvuus injektointiin.

Opinnäytetyön aikana opin paljon Java-sovelluksien tietokantayhteys työkaluista ja tietokannoista. Voin jatkossa käyttää opittuja taitojani työpaikallani, josta opinnäytetyön aihekin oli peräisin. Opinnäytetyö suoritettiin kuitenkin täysin itsenäisesti, salassapito velvollisuuksien vuoksi.

Opinnäytetyön tuloksia ja suosituksia voidaan hyödyntää uusien monimutkaisten Java-sovelluksien tietokanta hallintajärjestelmien ja työkalujen valinnassa sekä uusien ja vanhojen sovellusten suorituskyvyn optimoimiseen.

LÄHTEET

Flanagan, D. & Evans, B. 2014. Java in a Nutshell. 6. Yhdysvallat: O'Reilly Media.

Niemeyer, P. & Leuck, D. 2013. Learning Java. 4. Yhdysvallat: O'Reilly Media.

Bauer, C. & King, G. & Gregory, G. 2013. Java Persistence with Hibernate. 2. Yhdysvallat: Manning Publications.

Mihalcea, V. 2016. High-Performance Java Persistence. Romania: Vlad Mihalcea.

Getting Started with JavaFX. 2019. Oracle. Luettu 30.3.2019. https://docs.oracle.com/javase/8/javafx/get-started-tutorial/

Introduction to FXML. 2012. Oracle. Luettu 30.3.2019. https://docs.oracle.com/javafx/2/api/javafx/fxml/doc-files/introduction_to_fxml.html

What is MySQL. 2019. Oracle. Luettu 5.4.2019. https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html

Introduction to Oracle Database. 2015. Oracle. Luettu 5.4.2019. https://docs.oracle.com/cd/E11882_01/server.112/e40540/intro.htm#CNCPT88782

What is PostgreSQL. 2019. PostgreSQL. Luettu 5.4.2019. https://www.postgresql.org/about/

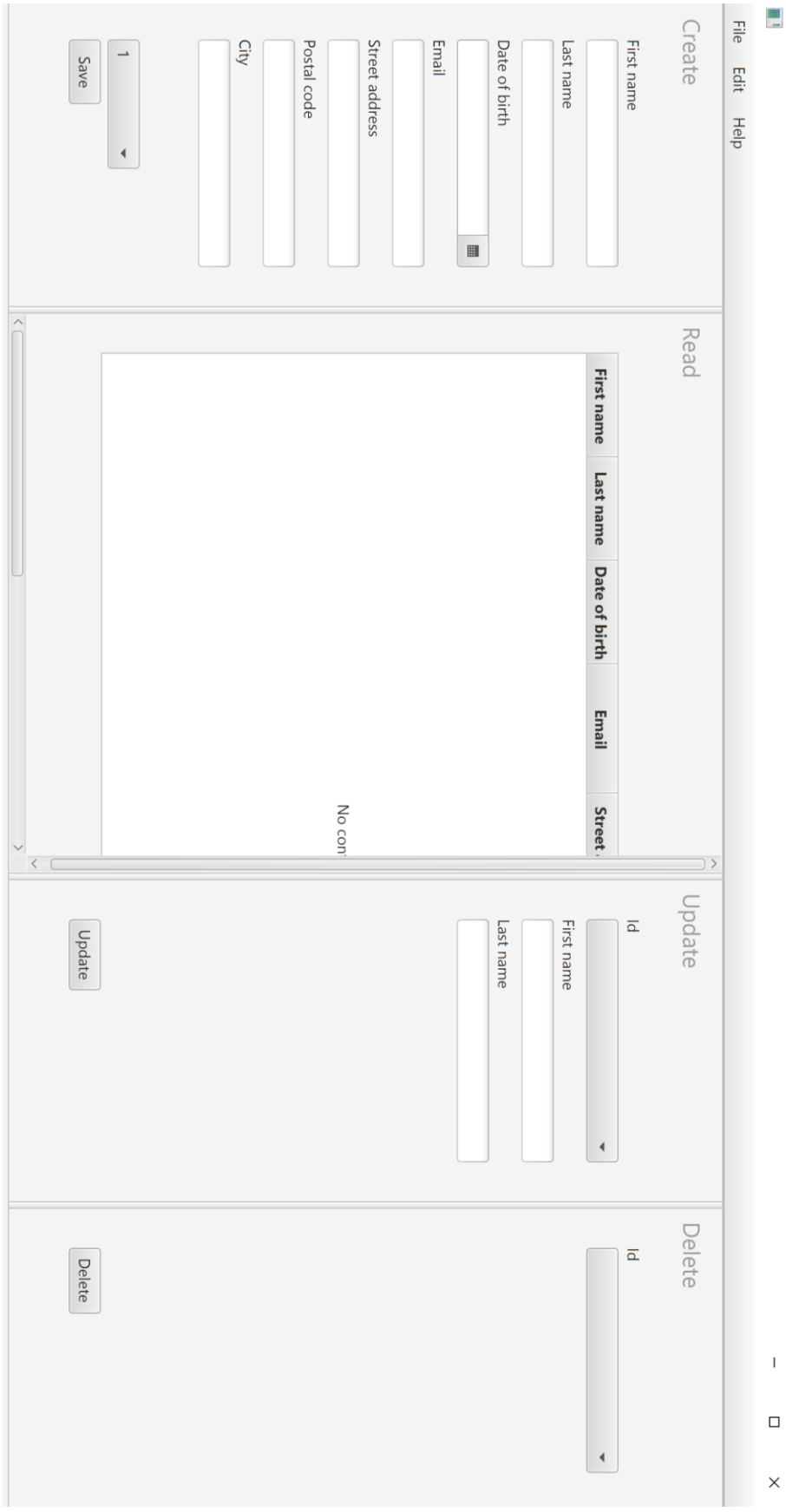
What is MongoDB. 2019. MongoDB. Luettu 6.4.2019. https://www.mongodb.com/what-is-mongodb

What is NoSQL. AWS. Luettu 30.3.2019. https://aws.amazon.com/nosql/

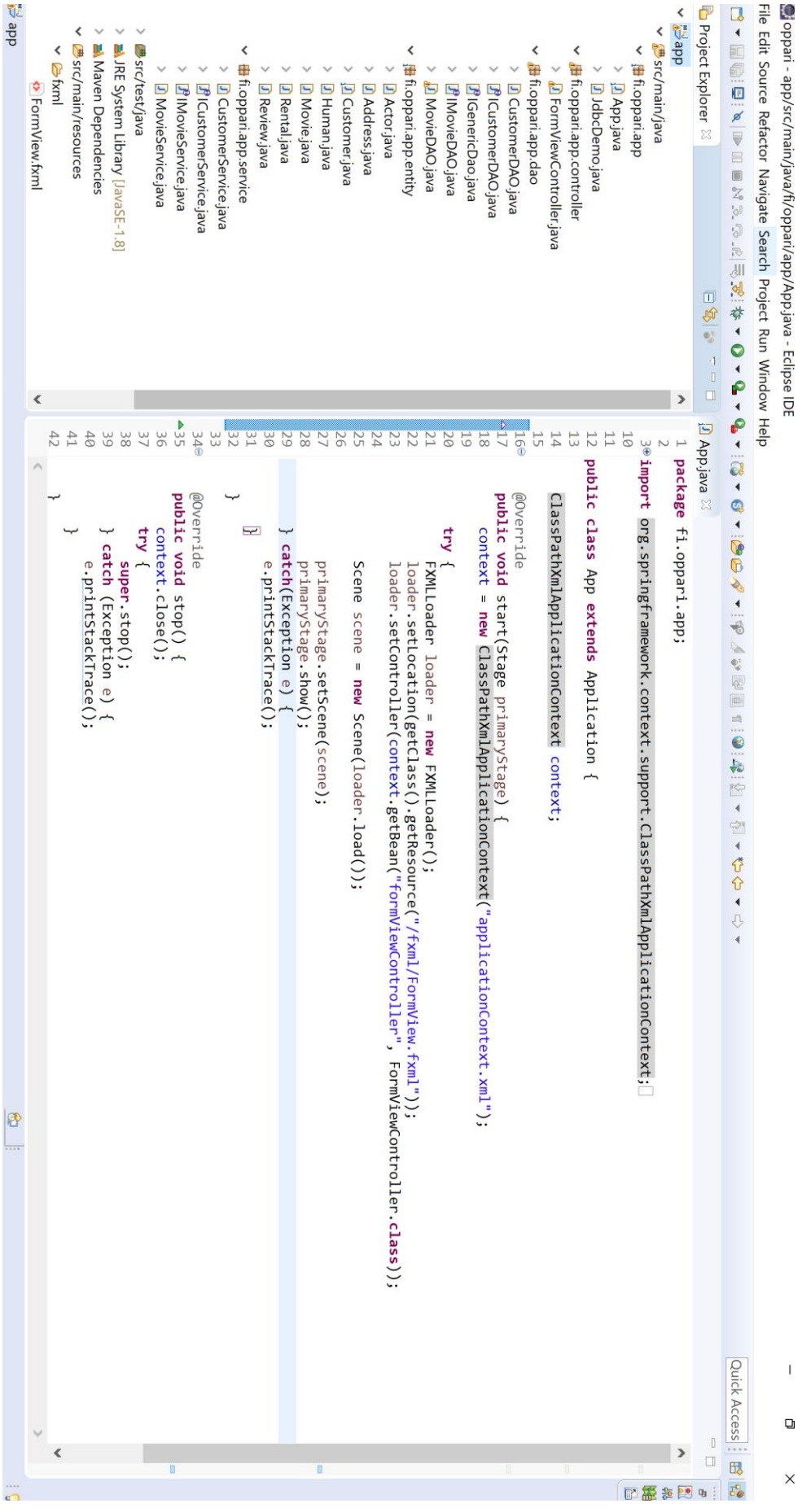
Mihalcea, V. 2019. Hibernate performance tuning tips. Luettu 26.4.2019. https://vladmihalcea.com/hibernate-performance-tuning-tips/

LIITTEET

Liite 1. Esimerkkisovelluksen UI



Liite 2. Eclipsen yleisnäkymä



Liite 3. EER-kaavio