Medicinska dijagnostika – JPA

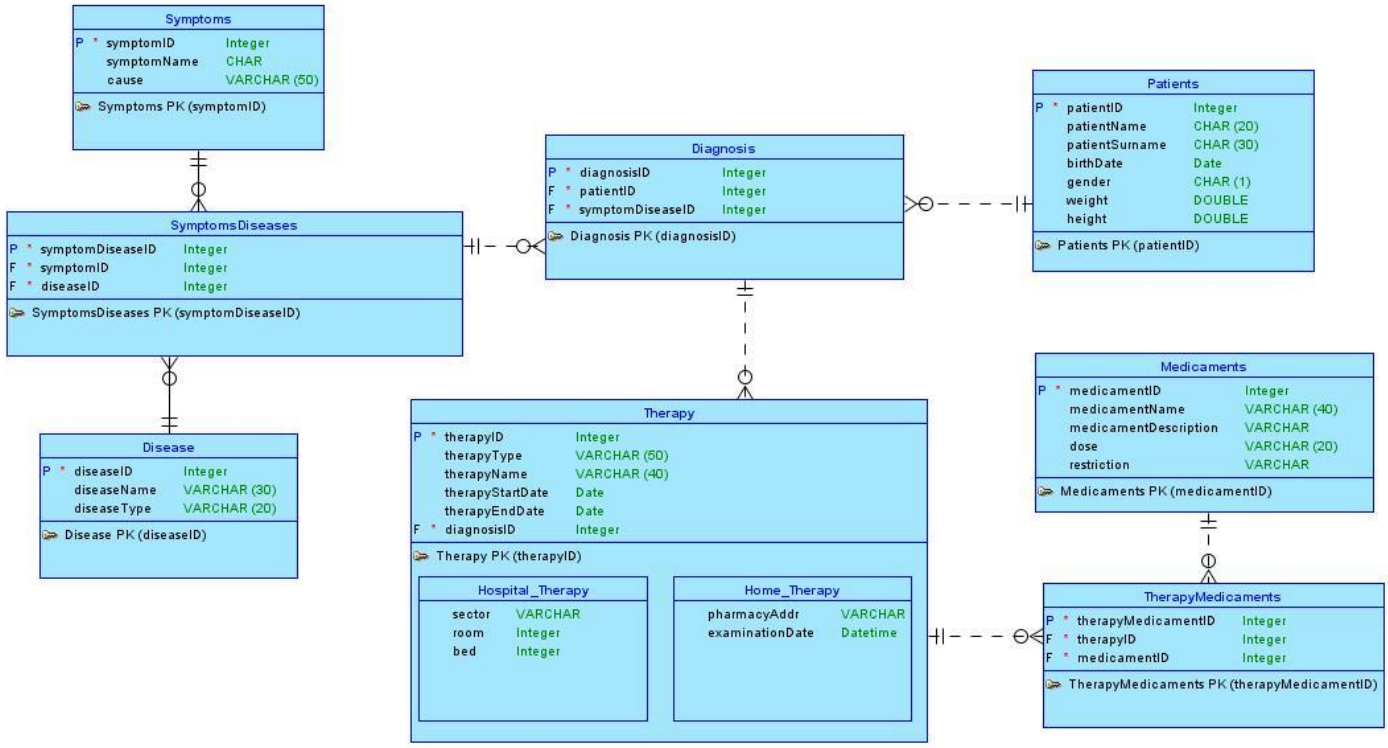
1. Uvod

Projekat predstavlja sistem za realizaciju Medicinske dijagnostike koga čine podaci o simptomima, bolestima, pacijentima, njihovim dijagnozama i terapijama sa prepisanim lekovima. Dijagnoze se ne mogu kreirati dok se ne zavedu podaci o pacijentu, kao i simptomima i bolestima. Takođe terapije ne mogu postojati dok se ne uradi bar jedna dijagnoza nekom pacijentu i dokle god nema lekova za određenu terapiju.

Pravila nalažu da se jedan simptom može naći kod više bolesti, kao i da jedna bolest može imati više simptoma, pa iz tog razloga imamo međuentitet za simptome i bolesti.
Takođe jedan pacijent može imati više dijagnoza i za jednu dijagnozu može se odrediti više terapija.
Jedna terapija može uključiti više lekova, takođe se i jedan lek može uključiti u više terapija.

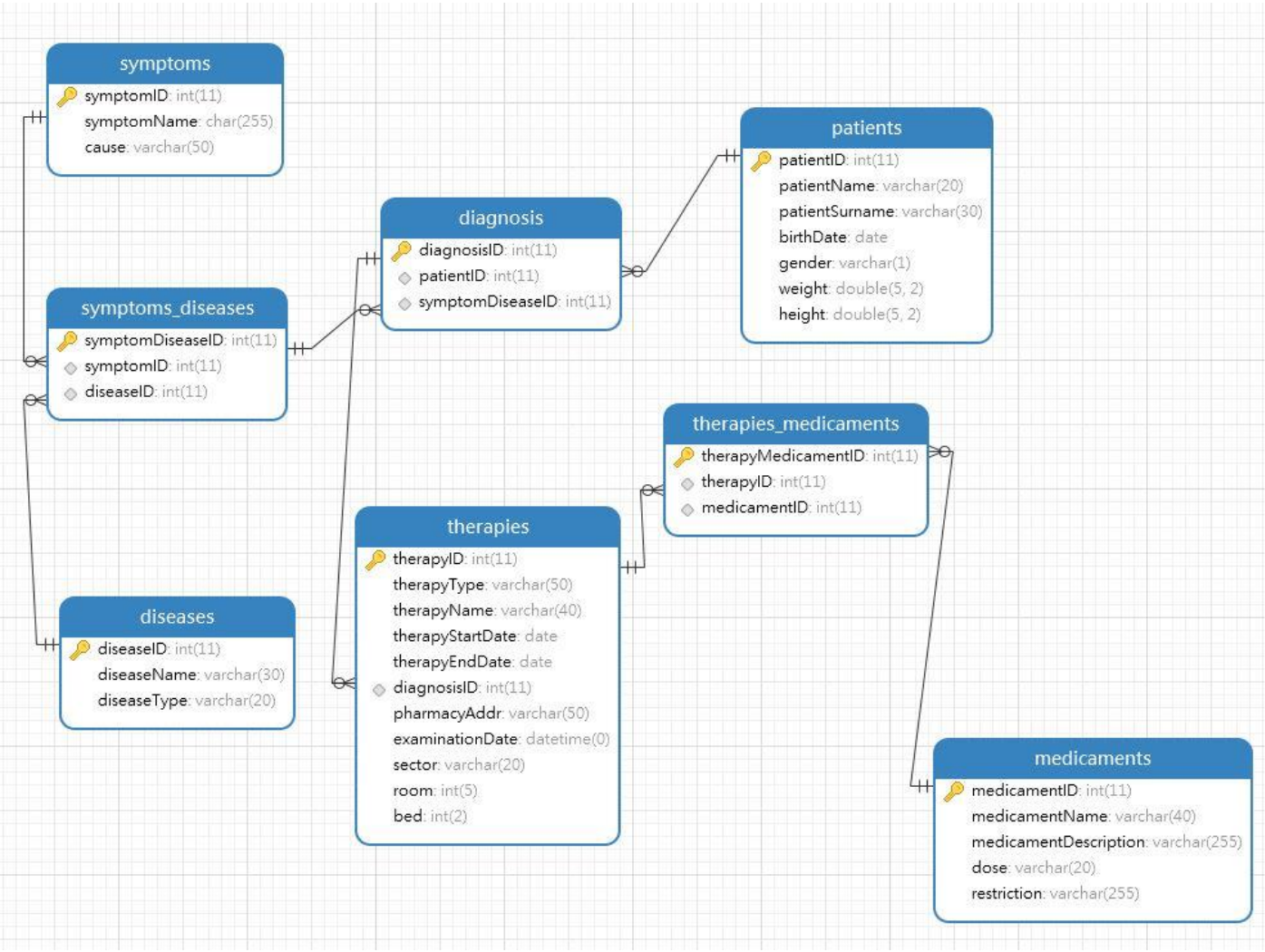
2. ERD Dijagram

Na dijagramu ispod prikazan je Entity-Relationship Diagram gde se može videti logički model sistema za Medicinsku dijagnostiku. Veza M:N između entiteta Bolesti - Simptoma, Terapija - Lekova je rešena uvođenjem međuentiteta kako bi vezu M:N transformisali u dve veze 1:N i M:1. Imamo prikaz generalizacije kod eniteta Terapija koja je generalizovana za podentitete Kućna terapija i Bolnička terapija. Ova generalizacija će biti rešena SINGLE_TABLE metodom gde su u fizičkom modelu svi podaci u jednoj tabeli. U logičkom kao i u OOP modelu imamo dve podklase koje nasleđuju osobine glavne apstraktne klase i sadrže samo svoje specijalne atribute.
Mapiranje veza u entitetu Terapija je rešena u glavnom entitetu. A diskriminator za specijalizaciju će prema svojoj vrednosti odrediti kog je specijalnog tipa konkretna Terapija.
Kada se reše veze M:N, vidimo da su sve veze na dijagramu 1:N tipa zbog navedenih poslovnih pravila.

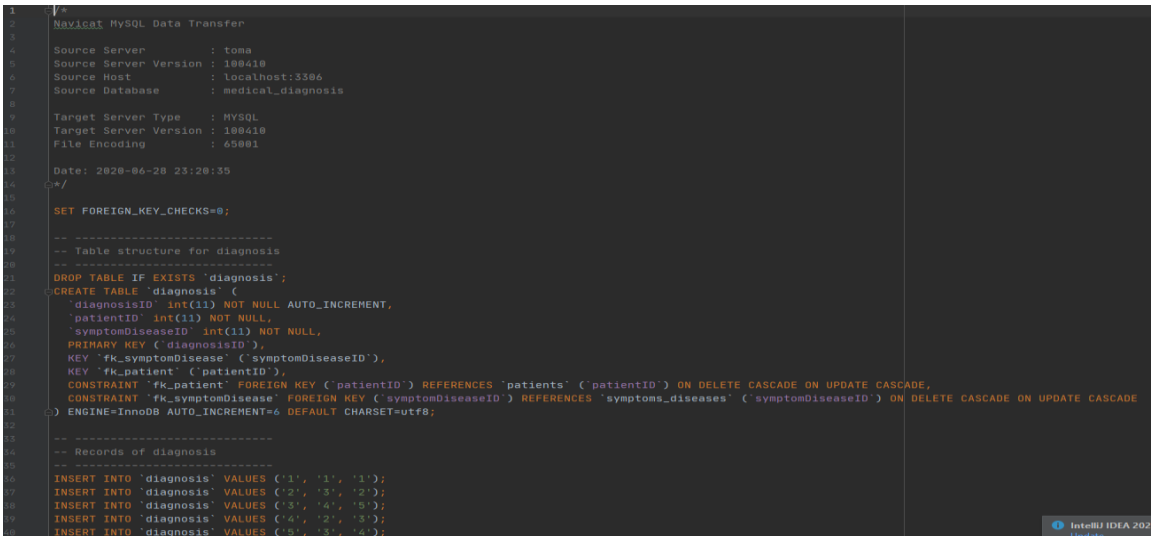


3. Fizički model

Na modelu ispod vidimo prikaz fizičkog modela baze gde su entiteti prevedeni u tabele i koji se ne razlikuje mnogo od logičkog modela. Jedina bitna razlika je što je generalizacija vezana za entitet Terapija rešena **SINGLE_TABLE** metodom gde imamo jednu tabelu u kojoj su **nullable** polja oba entiteta kao i DiscriminatorColumn – therapyType koji može imati vrednosti Hospital ili Home kako bi se znalo koja će dodatna polja biti popunjena a koja imati vrednost null.



Ispod je prikaz sadržaja sql skripte kojim se generiše fizički model baze podataka.



```
/*
Navicat MySQL Data Transfer

Source Server         : toma
Source Server Version : 100410
Source Host           : localhost:3306
Source Database       : medical_diagnosis

Target Server Type    : MYSQL
Target Server Version : 100410
File Encoding         : 65001

Date: 2020-06-28 23:20:35
*/

SET FOREIGN_KEY_CHECKS=0;

-- ----------------------------
-- Table structure for diagnosis
-- ----------------------------
DROP TABLE IF EXISTS `diagnosis`;
CREATE TABLE `diagnosis` (
  `diagnosisID` int(11) NOT NULL AUTO_INCREMENT,
  `patientID` int(11) NOT NULL,
  `symptomDiseaseID` int(11) NOT NULL,
  PRIMARY KEY (`diagnosisID`),
  KEY `fk_symptomDisease` (`symptomDiseaseID`),
  KEY `fk_patient` (`patientID`),
  CONSTRAINT `fk_patient` FOREIGN KEY (`patientID`) REFERENCES `patients` (`patientID`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `fk_symptomDisease` FOREIGN KEY (`symptomDiseaseID`) REFERENCES `symptoms_diseases` (`symptomDiseaseID`) ON DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=utf8;

-- ----------------------------
-- Records of diagnosis
-- ----------------------------
INSERT INTO `diagnosis` VALUES ('1', '1', '1');
INSERT INTO `diagnosis` VALUES ('2', '3', '2');
INSERT INTO `diagnosis` VALUES ('3', '4', '5');
INSERT INTO `diagnosis` VALUES ('4', '2', '3');
INSERT INTO `diagnosis` VALUES ('5', '3', '4');
```

4. Podešavanje razvojnog okruženja

- DBMS koji se koristi je MySQL gde je kreirana šema medical_diagnosis i koja je povezana sa JPA projektom preko .jar biblioteke konektora **mysql-connector-java-8.0.18.jar.**
- Provajder je Eclipse koji je preuzet preko Maven-a i naveden je u persistence.xml fajlu pod tagom **<provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>**
- Za razvoj projekta se koristi **IDEA IntelliJ 2020** okruženje.
- Potrebno je bilo setovati SDK za Java 14 verziju, kao i biblioteke potrebne za JPA poput javax.persistence, mysql-connector, org.eclipse.persistence.jpa i ostale. Takođe unutar src direktorijuma je bilo potrebno napraviti META-INF folder gde se nalazi persistence.xml kao JPA konfiguracuioni fajl. Takođe je bilo potrebno napraviti model folder sa svim definisanim klasama kao JPA entitetima i ui folder gde se nalazi glavna upravljačka klasa sa kojom se može vršiti interakcija preko konzole.

5. Klase sa relacijama i persistence.xml fajl

Persistence.xml fajl je prikazan ispod sa definisanim provajderom, klasama i jdbc paramaterima.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">

    <persistence-unit name="medical_diagnosis" transaction-type="RESOURCE_LOCAL">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>

        <class>model.Symptom</class>
        <class>model.Disease</class>
        <class>model.Patient</class>
        <class>model.Medicament</class>
        <class>model.Diagnose</class>
        <class>model.SymptomDisease</class>
        <class>model.TherapyMedicament</class>
        <class>model.Therapy</class>
        <class>model.HomeTherapy</class>
        <class>model.HospitalTherapy</class>

        <properties>
            <property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>
            <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/medical_diagnosis" />

            <property name="javax.persistence.jdbc.user" value="root"/>
            <property name="javax.persistence.jdbc.password" value=""/>

            <property name="eclipselink.Logging.Level" value="ALL"/>
            <property name="eclipselink.ddl-generation" value="create-tables"/>
            <property name="eclipselink.ddl-generation.output-mode" value="database"/>
        </properties>
    </persistence-unit>
</persistence>
```

Prikaz klase Pacijent je ispod. Ona je definisana kao JPA perzistentan entitet i povezana je sa tabelom patients. Takođe vidimo nekoliko Named upita definisanih iznad klase.

Svaka klasa kao enitet mora da implementira interfejs Serializable jer se u Entity Manager vrši serijalizacija entiteta kako bi se čuvale instance. Za id entiteta smo stavili da je GenerationType. IDENTITY što znači da korisitimo AUTO_INCREMENT mehanizam generisanja vrednosti primarnog ključa. Tako je i za sve ostale entitete definisano.

```java
package model;

import javax.persistence.*;
import java.io.Serializable;
import java.sql.Date;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "patients")
@NamedQueries({
    @NamedQuery(query = "SELECT p FROM Patient AS p WHERE p.birthDate <= :date", name="Patients older than given date"),
    @NamedQuery(query = "SELECT p FROM Patient AS p WHERE p.weight < (SELECT AVG(p.weight) FROM Patient as p)", name="Patients with lower weight than AVG"),
    @NamedQuery(query = "SELECT p FROM Patient AS p WHERE p.weight > :weight AND p.height < :height ORDER BY p.birthDate DESC", name="Sorted patients by birthDate wit
})
public class Patient implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "patientID")
    private int patientID;
    private String patientName, patientSurname;
    private Date birthDate;
    private char gender;
    private double weight, height;

    @OneToMany(mappedBy = "patient")
    private Set<Diagnose> diagnosis = new HashSet<~>();

    public Patient() {
    }

    public int getPatientID() { return patientID; }

    public void setPatientID(int patientID) { this.patientID = patientID; }

    public String getPatientName() { return patientName; }

    public void setPatientName(String patientName) { this.patientName = patientName; }

    public String getPatientSurname() { return patientSurname; }
```

Prikaz klase Disease koja takođe sadrži jedan Named upit i sve ostalo je na isti način definisano. Takođe imamo i mapiranu relaciju 1:N sa međuentitetom symptomsDiseases koja kao atribut ima HashSet strukturu podataka za brz pristup.

```java
package model;

import javax.persistence.*;
import java.io.Serializable;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "diseases")
@NamedQuery(query = "SELECT d FROM Disease d WHERE d.diseaseType = :type", name = "Find disease by type")
public class Disease  implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "diseaseID")
    private int diseaseID;
    private String diseaseName, diseaseType;

    @OneToMany(mappedBy = "disease")
    private Set<SymptomDisease> symptomsDiseases = new HashSet<~>();

    public Disease() {
    }

    public int getDiseaseID() { return diseaseID; }

    public void setDiseaseID(int diseaseID) { this.diseaseID = diseaseID; }

    public String getDiseaseName() { return diseaseName; }

    public void setDiseaseName(String diseaseName) { this.diseaseName = diseaseName; }

    public String getDiseaseType() { return diseaseType; }

    public void setDiseaseType(String diseaseType) { this.diseaseType = diseaseType; }

    public Set<SymptomDisease> getSymptomsDiseases() { return symptomsDiseases; }

    public void setSymptomsDiseases(Set<SymptomDisease> symptomsDiseases) { this.symptomsDiseases = symptomsDiseases; }

    @Override
    public String toString() {
```

Prikaz **Therapy** klase koja je apstraktna i generalizacija za klase HomeTherapy i HospitalTherapy. Strategija generalizacije je **SINGLE_TABLE** i koristi se therapyType kao diskriminatorska kolona. Takođe tabela ima vezu N:1 sa Dijagnozama jer se više terapija može naći u jendoj dijagnozi. S druge strane ima vezu 1:N sa međuentitetom sa medikamentima jer se više lekova može koristiti za jednu terapiju i zato se ovde takođe koristi HashSet **therapyMedicament** objekata.

```java
package model;

import javax.persistence.*;
import java.io.Serializable;
import java.sql.Date;
import java.util.HashSet;
import java.util.Set;

@Entity
@Table(name = "therapies")
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "therapyType")
public abstract class Therapy implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "therapyID")
    private int therapyID;
    private String therapyName;
    private Date therapyStartDate, therapyEndDate;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "diagnosisID")
    private Diagnose diagnose;

    @OneToMany(mappedBy = "therapy")
    private Set<TherapyMedicament> therapyMedicaments = new HashSet<>();

    public Therapy() {
    }

    public int getTherapyID() { return therapyID; }

    public void setTherapyID(int therapyID) { this.therapyID = therapyID; }

    public String getTherapyName() { return therapyName; }

    public void setTherapyName(String therapyName) { this.therapyName = therapyName; }

    public Date getTherapyStartDate() { return therapyStartDate; }

    public void setTherapyStartDate(Date therapyStartDate) { this.therapyStartDate = therapyStartDate; }
```

Prikaz HomeTherapy kao podklase Therapy koju i nasleđuje za vrednost Home diskriminatorske kolone. Dodata su nova polja na sve nasleđene podatke, međutim nasleđenim podacima se mora pristupati preko gettera jer su privatnog tipa podaci.

```java
package model;

import javax.persistence.DiscriminatorValue;
import javax.persistence.Entity;
import java.io.Serializable;
import java.sql.Timestamp;

@Entity
@DiscriminatorValue(value="Home")
public class HomeTherapy extends Therapy implements Serializable {
    private static final long serialVersionUID = 1L;
    private Timestamp examinationDate;
    private String pharmacyAddr;

    public HomeTherapy() {
    }

    public Timestamp getExaminationDate() { return examinationDate; }

    public void setExaminationDate(Timestamp examinationDate) { this.examinationDate = examinationDate; }

    public String getPharmacyAddr() { return pharmacyAddr; }

    public void setPharmacyAddr(String pharmacyAddr) { this.pharmacyAddr = pharmacyAddr; }

    @Override
    public String toString() {
        return "HomeTherapy{" +
                "therapyID=" + this.getTherapyID() +
                ", therapyName='" + this.getTherapyName() + '\'' +
                ", therapyStartDate=" + this.getTherapyStartDate() +
                ", therapyEndDate=" + this.getTherapyEndDate() +
                ", diagnosisID=" + this.getDiagnose().getDiagnosisID() +
                ", examinationDate=" + examinationDate +
                ", pharmacyAddr='" + pharmacyAddr + '\'' +
                '}';
    }
}
```

Ispod je prikaz glavne klase unutar ui foldera. Kreirali smo EntityManagerFactory preko koga smo kreirali EntityManager pomoću kog upravljamo transakcijama, pravimo upite, upravljamo radom entiteta...

Ispod toga je definicija pravljenja nekoliko vrsta upita kao što su Plain Query, TypedQuery, NamedQuery.

```java
package ui;
import model.*;

import javax.persistence.*;
import javax.persistence.criteria.*;
import java.sql.Date;
import java.util.List;

public class Main {
    public static void main(String[] args) {
        EntityManagerFactory emf = Persistence.createEntityManagerFactory( persistenceUnitName: "medical_diagnosis");
        EntityManager em = emf.createEntityManager();
        EntityTransaction tx = em.getTransaction();
        tx.begin();

        // Plain Query
        System.out.println("\nBelow is the result of Plain query which retrieves all patients:");
        List<Patient> patients = em.createQuery( s: "SELECT p FROM Patient as p ").getResultList();
        for(Patient patient: patients) {
            System.out.println(patient.toString());
        }

        // TypedQuery
        System.out.println("\nBelow is the result of Typed Query which retrieves all sypmtoms which name starts with 'Bol' (pain):");
        TypedQuery<Symptom> typedQuery = em.createQuery( s: "SELECT s FROM Symptom s WHERE s.symptomName LIKE 'Bol%'", Symptom.class);
        List<Symptom> symptomsWithPain = typedQuery.getResultList();
        for(Symptom symptom: symptomsWithPain) {
            System.out.println(symptom.toString());
        }

        // NamedQuery
        System.out.println("\nBelow is the result of Named Query which retrieves all diseases which type is Stomach:");
        Query query = em.createNamedQuery( s: "Find disease by type");
        query.setParameter( s: "type",  o: "Stomacno");
        List<Disease> diseasesStomach = query.getResultList();
        for(Disease disease: diseasesStomach){
            System.out.println(disease.toString());
        }

        // NamedQueries
        System.out.println("\nBelow is the result of Named Queries which retrieves specific patients:");

        Query query1 = em.createNamedQuery( s: "Patients older than given date");
```

Na slici ispod vidimo prikaz joše nekoliko načina kreiranja upita. Najbolji način i najlakši za debagovanje i održavanje Criteria API Query gde imamo 2 primera. Na kraju je prikazan i jedan složeniji upit sa INNER JOIN, WHERE uslovima i ORDER BY sortiranjem za entitete Pacijenti i Dijagnoze.

```java
        // Criteria API Query
        CriteriaBuilder cb = em.getCriteriaBuilder();
        System.out.println("\nBelow are the results of Criteria API Queries which retrieves specific patients:");
        CriteriaQuery<Object> criteriaQuery1 = cb.createQuery();
        Root<Medicament> from1 = criteriaQuery1.from(Medicament.class);
        System.out.println("\nSelect all medicaments:");
        CriteriaQuery<Object> select1 = criteriaQuery1.select(from1);
        TypedQuery<Object> typedQuery1 = em.createQuery(select1);
        List<Object> medicaments = typedQuery1.getResultList();
        for(Object obj: medicaments) {
            Medicament medicament = (Medicament) obj;
            System.out.println(medicament.toString());
        }

        CriteriaQuery<Object> criteriaQuery2 = cb.createQuery();
        Root<Therapy> from2 = criteriaQuery2.from(Therapy.class);
        System.out.println("\nSelect all therapies ordered by therapyStartDate descending:");
        CriteriaQuery<Object> select2 = criteriaQuery1.select(from2);
        select1.orderBy(cb.desc(from2.get("therapyStartDate")));
        TypedQuery<Object> typedQuery2 = em.createQuery(select2);
        List<Object> therapies = typedQuery2.getResultList();
        for(Object obj: therapies) {
            Therapy therapy = (Therapy) obj;
            System.out.println(therapy.toString());
        }

        // Complex Query with joins, etc.
        System.out.println("\nBelow are the results of the complex queries with joins:\n");
        TypedQuery<Patient> complexQuery1 = em.createQuery( s: "SELECT DISTINCT p FROM Diagnose d INNER JOIN d.patient p WHERE d.patient.weight < 80.0 ORDER BY d.patient.height DESC",
                Patient.class);
        List<Patient> patientsJoinDiagnoses = complexQuery1.getResultList();
        for(Patient patient: patientsJoinDiagnoses){
            System.out.println(patient.toString());
            System.out.println("Patients diagnoses:");
            for(Diagnose diagnose: patient.getDiagnosis()){
                System.out.println(diagnose.toString());
            }
            System.out.println();
        }

        tx.commit();
        em.close();
```

6. Prikaz konzole i baze

Prikaz rezultata svih pacijenata običnog upita.

```
Below is the result of Plain query which retrieves all patients:
Patient{patientID=1, patientName='Toma', patientSurname='Joksimovic', birthDate=1997-09-13, gender=M, weight=90.0, height=186.0}
Patient{patientID=2, patientName='Mirko', patientSurname='Mirkovic', birthDate=1995-10-23, gender=M, weight=84.0, height=187.0}
Patient{patientID=3, patientName='Ana', patientSurname='Dinkic', birthDate=1999-03-12, gender=F, weight=69.0, height=174.0}
Patient{patientID=4, patientName='Marija', patientSurname='Stankovic', birthDate=1983-08-30, gender=F, weight=71.0, height=180.0}
```

Prikaz rezultata svih simptoma koji počinju sa „Bol“ nazivom za Typed upit.

```
Below is the result of Typed Query which retrieves all sypmtoms which name starts with 'Bol' (pain):
Symptom{symptomID=2, symptomName='Bol u glavi', cause='Preveliko izlaganje suncu'}
Symptom{symptomID=3, symptomName='Bol u grlu', cause='Konzumacija hladne hrane i pica'}
```

Prikaz rezultata bolesti tipa Stomačnog oboljenja korišćenjem Named upita.

```
Below is the result of Named Query which retrieves all diseases which type is Stomach:
Disease{diseaseID=3, diseaseName='Stomacni virus', diseaseType='Stomacno'}
```

Prikaz rezultata kolekcije Named upita.

```
Below is the result of Named Queries which retrieves specific patients:

Patients older than 1998-04-24:
Patient{patientID=1, patientName='Toma', patientSurname='Joksimovic', birthDate=1997-09-13, gender=M, weight=90.0, height=186.0}
Patient{patientID=2, patientName='Mirko', patientSurname='Mirkovic', birthDate=1995-10-23, gender=M, weight=84.0, height=187.0}
Patient{patientID=4, patientName='Marija', patientSurname='Stankovic', birthDate=1983-08-30, gender=F, weight=71.0, height=180.0}

Patients with lower weight than AVG:
Patient{patientID=3, patientName='Ana', patientSurname='Dinkic', birthDate=1999-03-12, gender=F, weight=69.0, height=174.0}
Patient{patientID=4, patientName='Marija', patientSurname='Stankovic', birthDate=1983-08-30, gender=F, weight=71.0, height=180.0}

Sorted patients by birthDate with weight gt given and height lt given
Patient{patientID=1, patientName='Toma', patientSurname='Joksimovic', birthDate=1997-09-13, gender=M, weight=90.0, height=186.0}
Patient{patientID=4, patientName='Marija', patientSurname='Stankovic', birthDate=1983-08-30, gender=F, weight=71.0, height=180.0}
```

Prikaz nekoliko rezultata primenom Criteria API Query objektom CriteriaBuilder.

```
Below are the results of Criteria API Queries which retrieves specific patients:

Select all medicaments:
Medicament{medicamentID=1, medicamentName='Kafetin', medicamentDescription='Deluje protiv bola u glavi', dose='1*250mg na dan', restriction='Ne mesati sa alkoholom'}
Medicament{medicamentID=2, medicamentName='Probiotik', medicamentDescription='Deluje protiv bola u stomaku', dose='3*kapsula na dan', restriction='Ne konzumirati na prazan stomak'}
Medicament{medicamentID=3, medicamentName='Strepsils', medicamentDescription='Ublazava bol u grlu', dose='2 tablete na dan', restriction='Ne kozumirati na prazan stomak'}
Medicament{medicamentID=4, medicamentName='Caj od zalfije', medicamentDescription='Za upalu grla, groznicu, stomacne tegobe', dose='4-5 solji na dan', restriction='Ne konzumirati pred spavanje'}
Medicament{medicamentID=5, medicamentName='Hemomicin', medicamentDescription='Za virusne infkecije', dose='1*500mg na dan*6', restriction='Ne konzumirati na prazan stomak i sa alkoholom'}

Select all therapies ordered by therapyStartDate descending:
HospitalTherapy{therapyID=2, therapyName='Inhalacija', therapyStartDate=2020-06-20, therapyEndDate=2020-06-29, diagnosisID=5, sector='5', room=2, bed=9}
HomeTherapy{therapyID=1, therapyName='Lecenje prehlade', therapyStartDate=2020-06-11, therapyEndDate=2020-06-26, diagnosisID=1, examinationDate=2020-06-24 15:57:31.0, pharmacyAddr='Masarikova 10'}
```

Prikaz složenog upita korišćenjem INNER JOIN metoda spajanja entiteta Patient i Diagnose.

```
Below are the results of the complex queries with joins:

Patient{patientID=4, patientName='Marija', patientSurname='Stankovic', birthDate=1983-08-30, gender=F, weight=71.0, height=180.0}
Patients diagnoses:
Diagnose{diagnosisID=3, patientID=4, symptomDisease=5}

Patient{patientID=3, patientName='Ana', patientSurname='Dinkic', birthDate=1999-03-12, gender=F, weight=69.0, height=174.0}
Patients diagnoses:
Diagnose{diagnosisID=5, patientID=3, symptomDisease=4}
Diagnose{diagnosisID=2, patientID=3, symptomDisease=2}
```

Prikaz nekoliko tabela u bazi podataka:

Terapije

therapyID	therapyType	therapyName	therapyStartDate	therapyEndDate	diagnosisID	pharmacyAddr	examinationDate	sector	room	bed
1	Home	Lecenje prehlade	2020-06-11	2020-06-26	1	Masarikova 10	2020-06-24 15:57:31		(Null)	(Null)
2	Hospital	Inhalacija	2020-06-20	2020-06-29	5	(Null)	(Null)	5	2	9

Pacijenti

patientID	patientName	patientSurname	birthDate	gender	weight	height
1	Toma	Joksimovic	1997-09-13	M	90	186
2	Mirko	Mirkovic	1995-10-23	M	84	187
3	Ana	Dinkic	1999-03-12	F	69	174
4	Marija	Stankovic	1983-08-30	F	71	180

Dijagnoze

diagnosisID	patientID	symptomDiseaseID
1	1	1
2	3	2
3	4	5
4	2	3
5	3	4

Medikamenti

medicamentID	medicamentName	medicamentDescription	dose	restriction
1	Kafetin	Deluje protiv bola u glavi	1*250mg na dan	Ne mesati sa alkoholom
2	Probiotik	Deluje protiv bola u stomaku	3*kapsula na dan	Ne konzumirati na prazan stomak
3	Strepsils	Ublazava bol u grlu	2 tablete na dan	Ne kozumirati na prazan stomak
4	Caj od zalfije	Za upalu grla, groznicu, stomacne tegobe	4-5 solji na dan	Ne konzumirati pred spavanje
5	Hemomicin	Za virusne infkecije	1*500mg na dan*6	Ne konzumirati na prazan stomak i sa alkoholom

7. Zaključak

Ograničenja projekta su potrebno ekspertsko znanje kako bi se eventualno kreirao Ekspertski sistem ili implementirali algoritmi Mašinskog učenja na osnovu velikog broja raznovrsnih uzoraka (za koje bi bila potrebna velika količina različitih slučajeva) kako bi se donela dobra i što preciznija dijagnoza na osnovu simptoma pacijenta i odredila što efikasnija terapija.

Dodatno tehničko ograničenje je nepostajanje mogućnosti da se sistem implementira kao distribuiran koji bi radio u realnom vremenu i kao višenitna aplikacija koja komunicira sa udaljenim serverima. Za takav slučaj je potrebno bolje organizovati aplikaciju po Mikroservisnoj arhitekturi i slojevitije organizovati podatke sa interakciju sa bazom sa jasnim DAO i DTO fajlovima, kao i da kod bude lako održiv i da aplikacija može biti skalabilna.

8. Literatura

- Materijali sa predavanja/vežbi iz predmeta Praktikum – Napredno Softversko Inženjerstvo 2020
- https://www.baeldung.com/jpa-join-types
- https://thorben-janssen.com/jpql/
- https://stackoverflow.com/