# heJPA (Java Persistence API)

პრაქტიკული მაგალითების გაკეთება დავიწყოთ JPA სპეციფიკაციით. ანუ პირველ რიგში გამოვიყენოთ Hibernate-ის JPA ნაწილი, რომელიც Java EE-ს მიხედვით არის შექმნილი, არის პორტაბელური, სტანდარტიზირებული და კოდის ცვლილების გარეშეც ერგება სხვა JPA პროვაიდერს Hibernate-ის გარდა. თუმცა პრაქტიკულად აქ Hibernate-ის JPA რეალიზაციას გამოვიყნებეთ.

როგორც ვიცით, JPA-ს კლასები და ანოტაციები განთავსებულია javax.persistence.\* პაკეტებში, რომელიც არ მოჰყვება Java SE-ს. pom-ში უნდა დავამატოთ <artifactId>hibernate-core</artifactId>, ხოლო JPA-ს კონფიგურაციის ფაილი persistence.xml შევინახოთ პროექტის resources\META-INF დირექტორიაში.

სანამ ახალ პროექტს დავაგენერირებდეთ, შევქმნათ პირველი მაგალითისთვის საჭირო ცხრილი:

connect 'jdbc:derby://localhost:1527/sample';

**CREATE** **TABLE** Students **(**

student\_id INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY** **(START** **WITH** 1**,** **INCREMENT** **BY** 1**),**

name VARCHAR**(**100**)** **NOT** **NULL,**

email VARCHAR**(**100**),**

creation\_date TIMESTAMP**,**

**Unique** **(**name**)**

**);**

ახლა კი დავაგენერიროთ ახალი პროექტი: mvn archetype:generate -DgroupId=ge.ufc.jpa -DartifactId=jpa-hibernate-1 -Dversion=1.0 -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false

pom.xml

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<project xmlns=**"http://maven.apache.org/POM/4.0.0"** xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"** xsi:schemaLocation=**"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>

<modelVersion>**4.0.0**</modelVersion>

<groupId>**ge.ufc.pooling**</groupId>

<artifactId>**jpa-hibernate-1**</artifactId>

<version>**1.0**</version>

<name>**jpa-hibernate-1**</name>

<url>**http://www.ufc.ge**</url>

<properties>

<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>

<maven.compiler.release>**11**</maven.compiler.release>

<derby.version>**10.15.2.0**</derby.version>

<hibernate.version>**5.6.12.Final**</hibernate.version>

</properties>

<dependencies>

<dependency>

<groupId>**org.hibernate**</groupId>

<artifactId>**hibernate-core**</artifactId>

<version>**${hibernate.version}**</version>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbyclient**</artifactId>

<version>**${derby.version}**</version>

<scope>**runtime**</scope>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbytools**</artifactId>

<version>**${derby.version}**</version>

<scope>**runtime**</scope>

</dependency>

</dependencies>

<build>

<finalName>**jpaHibernate1**</finalName>

<plugins>

<plugin>

<artifactId>**maven-compiler-plugin**</artifactId>

<version>**3.8.1**</version>

</plugin>

<plugin>

<groupId>**org.apache.maven.plugins**</groupId>

<artifactId>**maven-eclipse-plugin**</artifactId>

<version>**2.10**</version>

<configuration>

<projectNameTemplate>

**[artifactId]-[version]**

</projectNameTemplate>

<wtpmanifest>**true**</wtpmanifest>

<wtpapplicationxml>**true**</wtpapplicationxml>

<wtpversion>**2.0**</wtpversion>

<manifest>

**${basedir}/src/main/resources/META-INF/MANIFEST.MF**

</manifest>

</configuration>

</plugin>

<plugin>

<artifactId>**maven-assembly-plugin**</artifactId>

<version>**3.4.2**</version>

<executions>

<execution>

<phase>**package**</phase>

<goals>

<goal>**single**</goal>

</goals>

<configuration>

<descriptorRefs>

<descriptorRef>**jar-with-dependencies**</descriptorRef>

</descriptorRefs>

<appendAssemblyId>**false**</appendAssemblyId>

<archive>

<manifest>

<mainClass>**ge.ufc.jpa.App**</mainClass>

</manifest>

</archive>

</configuration>

</execution>

</executions>

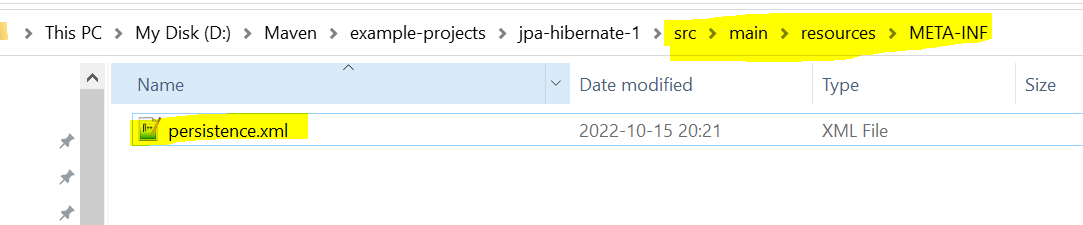
</plugin>

</plugins>

</build>

</project>

src\main\resources\META-INF\persistence.xml



<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<persistence xmlns=**"http://xmlns.jcp.org/xml/ns/persistence"** xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"** xsi:schemaLocation=**"http://xmlns.jcp.org/xml/ns/persistence**

**http://xmlns.jcp.org/xml/ns/persistence/persistence\_2\_2.xsd"** version=**"2.2"**>

<persistence-unit name=**"sampleDB"**>

<properties>

<property name=**"hibernate.connection.driver\_class"** value=**"org.apache.derby.jdbc.ClientDriver"**/>

<property name=**"hibernate.connection.url"** value=**"jdbc:derby://localhost:1527/sample"**/>

<property name=**"hibernate.dialect"** value=**"org.hibernate.dialect.DerbyTenSevenDialect"**/>

<property name=**"hibernate.show\_sql"** value=**"true"**/>

</properties>

</persistence-unit>

</persistence>

როგორც ხედავთ, აქ გავწერეთ ბაზის დრაივერის კლასი და კავშირის სტრიქონი. ასევე ჩავრთეთ show\_sql, რომ დაილოგოს Hibernate-ის მიერ დაგენერირებული SQL ბრძანებები და რაც მთავარია, Hibernate-მა რომ იცოდეს, რომელ ბაზასთან აქვს საქმე და რა სინტაქსის SQL-ით ემუშაოს (JPA-ს უკან JDBC დგას), აუცილებლად სჭირდება SQL დიალექტის გაწერა, რომელშიც გავწერეთ org.hibernate.dialect.DerbyTenSevenDialect მნიშვნელობა, ანუ დროის ამ მომენტისთვის Hibernate-ის მიერ დერბის დიალექტის უახლესი ვერსია შეესაბამება Derby 10.7 ვერსიას, მაგრამ ჩვენს 10.15 ვერსიაზეც უპრობლემოდ მუშაობს.

mvn clean package გავბილდოთ პროექტი და შევაიმპორტოთ ეკლიფსში mvn eclipse:eclipse

პროექტს დავამატოთ კლასები:

package ge**.**ufc**.**jpa**;**

**import** java**.**util**.**Date**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**GeneratedValue**;**

**import** javax**.**persistence**.**GenerationType**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**Table**;**

**import** javax**.**persistence**.**Temporal**;**

**import** javax**.**persistence**.**TemporalType**;**

@Entity

@Table**(**name **=** "Students"**)**

public class Student **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

@Column**(**name **=** "student\_id"**)**

private int id**;**

@Column**(**name **=** "name"**,** unique **=** **true)**

private String name**;**

@Column**(**name **=** "email"**)**

private String email**;**

@Temporal**(**TemporalType**.**TIMESTAMP**)**

@Column**(**name **=** "creation\_date"**)**

private Date creationDate**;**

public Student**(**String name**,** String email**)** **{**

**this.**name **=** name**;**

**this.**email **=** email**;**

**}**

public Student**()** **{**

**}**

public int getId**()** **{**

**return** id**;**

**}**

public void setId**(**int id**)** **{**

**this.**id **=** id**;**

**}**

public String getName**()** **{**

**return** name**;**

**}**

public void setName**(**String name**)** **{**

**this.**name **=** name**;**

**}**

public String getEmail**()** **{**

**return** email**;**

**}**

public void setEmail**(**String email**)** **{**

**this.**email **=** email**;**

**}**

public Date getCreationDate**()** **{**

**return** creationDate**;**

**}**

public void setCreationDate**(**Date creationDate**)** **{**

**this.**creationDate **=** creationDate**;**

**}**

@Override

public String toString**()** **{**

StringBuilder builder **=** **new** StringBuilder**();**

builder**.**append**(**"Student {id="**);**

builder**.**append**(**id**);**

builder**.**append**(**", name="**);**

builder**.**append**(**name**);**

builder**.**append**(**", email="**);**

builder**.**append**(**email**);**

builder**.**append**(**", created="**);**

builder**.**append**(**creationDate**);**

builder**.**append**(**"}"**);**

**return** builder**.**toString**();**

**}**

**}**

--------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** java**.**util**.**ArrayList**;**

**import** java**.**util**.**Date**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**persistence**.**EntityManager**;**

**import** javax**.**persistence**.**EntityManagerFactory**;**

**import** javax**.**persistence**.**Persistence**;**

**import** javax**.**persistence**.**Query**;**

**import** javax**.**persistence**.**TypedQuery**;**

public class App **{**

private static final EntityManagerFactory emf**;**

private static final EntityManager em**;**

private static final List**<**Student**>** students**;**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

emf **=** Persistence**.**createEntityManagerFactory**(**"sampleDB"**);**

em **=** emf**.**createEntityManager**();**

students **=** **new** ArrayList**<>();**

students**.**add**(new** Student**(**"Oliver"**,** "Oliver@gmail.com"**));**

students**.**add**(new** Student**(**"Lucas"**,** "Lucas@gmail.com"**));**

students**.**add**(new** Student**(**"Henry"**,** "Henry@yahoo.com"**));**

students**.**add**(new** Student**(**"Theodore"**,** "Th\_dr@live.com"**));**

students**.**add**(new** Student**(**"Sophia"**,** "sophia.sh@mail.com"**));**

students**.**add**(new** Student**(**"Isabella"**,** "Isabel@missioncollege.edu"**));**

students**.**add**(new** Student**(**"Mia"**,** "miamia@citruscollege.edu"**));**

students**.**add**(new** Student**(**"William"**,** "William@yahoo.com"**));**

**}**

public static void main**(**String**[]** args**)** **{**

**try** **{**

students**.**forEach**(**App**::**createStudent**);**

System**.**out**.**println**();**

findStudent**(**"Lucas"**);**

System**.**out**.**println**();**

A

searchStudent**(**"ia"**);**

System**.**out**.**println**();**

updateStudent**(**students**.**get**(**4**).**getId**(),** "sophia.sh@gmail.com"**);**

System**.**out**.**println**();**

deleteStudent**(**students**.**get**(**6**).**getId**());**

System**.**out**.**println**();**

listStudents**();**

System**.**out**.**println**();**

truncateStudent**();**

**}** **finally** **{**

**if** **(**em **!=** **null)** **{**

em**.**close**();**

**}**

**if** **(**emf **!=** **null)** **{**

emf**.**close**();**

**}**

**}**

**}**

private static void createStudent**(**Student student**)** **{**

student**.**setCreationDate**(new** Date**());**

em**.**getTransaction**().**begin**();**

em**.**persist**(**student**);**

em**.**getTransaction**().**commit**();**

System**.**out**.**println**(**"Added Student: " **+** student**);**

**}**

private static void findStudent**(**String name**)** **{**

**try** **{**

TypedQuery**<**Student**>** query **=** em**.**createQuery**(**"select s from " **+** Student**.**class**.**getName**()** **+** " s where s.name = :name"**,** Student**.**class**);**

query**.**setParameter**(**"name"**,** name**);**

Student student **=** query**.**getSingleResult**();**

System**.**out**.**println**(**"Your result---Find student:"**);**

System**.**out**.**println**(**student**);**

**}** **catch** **(**Exception ex**)** **{**

System**.**err**.**println**(**"Some exception occured: " **+** ex**.**getMessage**());**

**}**

**}**

private static void searchStudent**(**String name**)** **{**

**try** **{**

TypedQuery**<**Student**>** query **=** em**.**createQuery**(**

"select s from " **+** Student**.**class**.**getName**()** **+** " s where lower(s.name) like :search"**,** Student**.**class**);**

query**.**setParameter**(**"search"**,** "%" **+** name**.**toLowerCase**()** **+** "%"**);**

List**<**Student**>** list **=** query**.**getResultList**();**

**if** **(**list**.**size**()** **==** 0**)** **{**

System**.**out**.**println**(**"Nothing found where name like " **+** name**);**

**return;**

**}**

System**.**out**.**println**(**"Your result---Indexed students:"**);**

list**.**forEach**(**System**.**out**::**println**);**

**}** **catch** **(**Exception ex**)** **{**

System**.**err**.**println**(**"Some exception occured: " **+** ex**.**getMessage**());**

**}**

**}**

private static void updateStudent**(**int studentId**,** String newEmail**)** **{**

Student student **=** em**.**find**(**Student**.**class**,** studentId**);**

**if** **(**student **==** **null)** **{**

System**.**out**.**println**(**"No student was found with id " **+** studentId**);**

**}** **else** **{**

System**.**out**.**println**(**"Updating student " **+** student**);**

student**.**setEmail**(**newEmail**);**

em**.**getTransaction**().**begin**();**

em**.**persist**(**student**);**

em**.**getTransaction**().**commit**();**

System**.**out**.**println**(**"The student was updated successfully"**);**

**}**

**}**

private static void deleteStudent**(**int studentId**)** **{**

Student student **=** em**.**getReference**(**Student**.**class**,** studentId**);** //or find

**if** **(**student **==** **null)** **{**

System**.**out**.**println**(**"No student was found with id " **+** studentId**);**

**}** **else** **{**

System**.**out**.**println**(**"Deleting student " **+** student**);**

em**.**getTransaction**().**begin**();**

em**.**remove**(**student**);**

em**.**getTransaction**().**commit**();**

System**.**out**.**println**(**"The student was deleted successfully"**);**

**}**

**}**

private static void listStudents**()** **{**

Query query **=** em**.**createQuery**(**"select s from " **+** Student**.**class**.**getName**()** **+** " s order by s.id"**);**

List**<?>** students **=** query**.**getResultList**();**

System**.**out**.**println**(**"Your result---List students:"**);**

System**.**out**.**println**(**students**);**

System**.**out**.**println**(**String**.**valueOf**(**students**.**size**())** **+** " row(s) selected"**);**

**}**

private static void truncateStudent**()** **{**

**try** **{**

Query query **=** em**.**createQuery**(**"select s from " **+** Student**.**class**.**getName**()** **+** " s"**);**

@SuppressWarnings**(**"unchecked"**)**

List**<**Student**>** list **=** query**.**getResultList**();**

**if** **(**list**.**size**()** **==** 0**)** **{**

System**.**out**.**println**(**"Your list is already empty!"**);**

**return;**

**}**

em**.**getTransaction**().**begin**();**

list**.**forEach**(**em**::**remove**);**

em**.**getTransaction**().**commit**();**

System**.**out**.**println**(**"Your list is now successfully cleared"**);**

**}** **catch** **(**Exception ex**)** **{**

System**.**err**.**println**(**"Some exception occured: " **+** ex**.**getMessage**());**

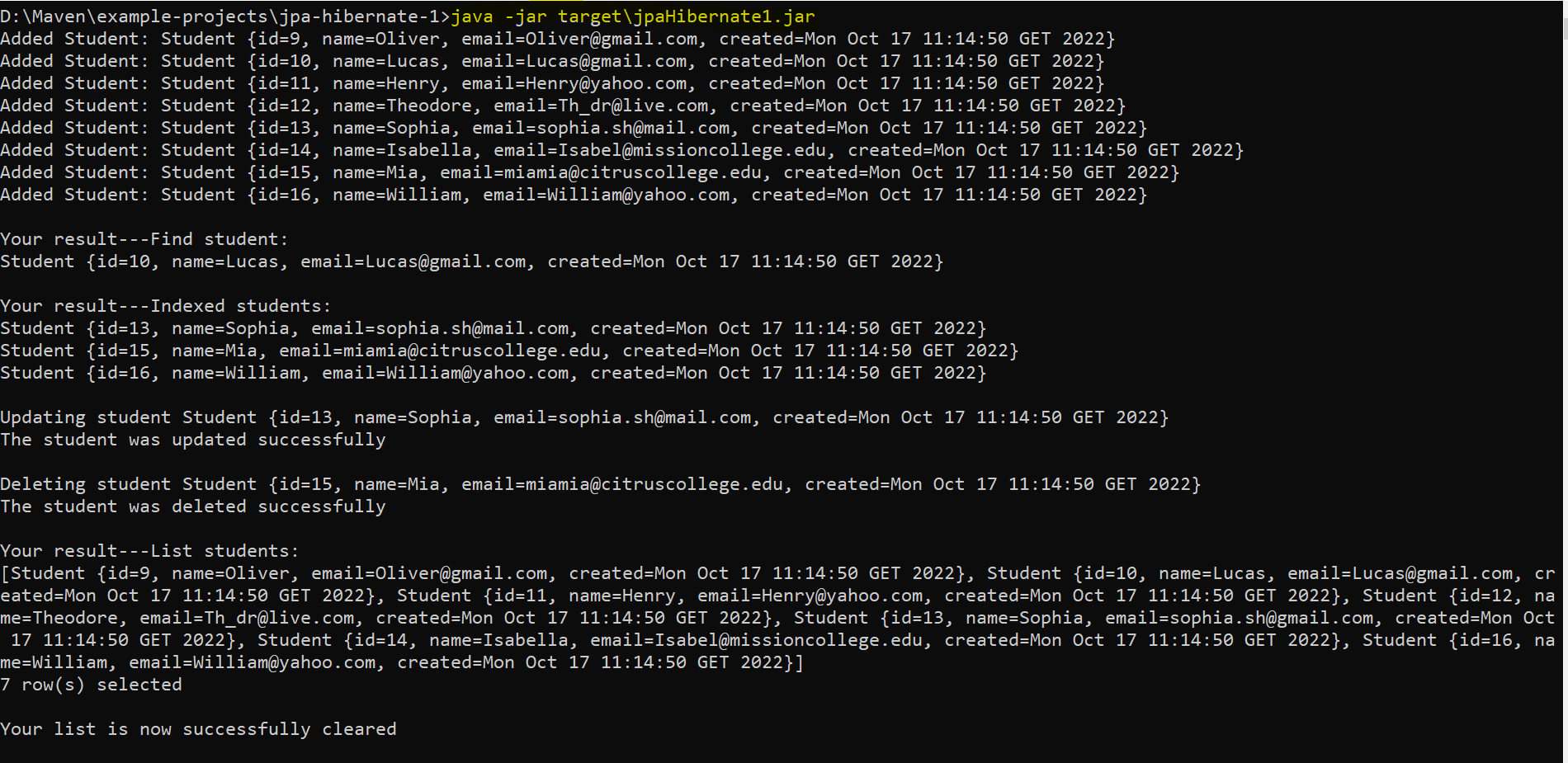
**}**

**}**

**}**

კიდევ ერთხელ გავბიდლოთ პროექტი mvn clean package, დროებით გავთიშოთ show\_sql და გავუშვათ: java -jar target\jpaHibernate1.jar

უნდა მივიღოთ მსგავსი შედეგი:



Student კლასში @Entity ანოტაცია აღნიშნავს, რომ ის არის მარტივი Entity (POJO) კლასი და დაკავშირიებულია ბაზის ცხრილთან. @Table-ში აღწერილია ცხრილის შესაბამისი სახელი ბაზაში, ხოლო @Column-ებში ცხრილი შესაბამისი ველების სახელები. @Id აღწერს ცხრილის პირველად გასაღებს, @GeneratedValue(strategy = GenerationType.IDENTITY) კი აღწერს, რომ ველის მნიშვნელობა ავტომატურად გენერირდება ბაზაში. @Column ანოტაციის unique = true გვეხმარება ცხრილის შესაბამის კლასში აღვწეროთ ისეთი ველი, რომელიც ცხრილის პირველადი გასაღები არ არის, მაგრამ უნიკალურია ცხრილის ფარგლებში. @Temporal(TemporalType.TIMESTAMP) აღწერს, რომ თარიღის ველში ცხრილში შენახულია როგორც თარიღი, ისე დრო.

Persistence.createEntityManagerFactory მეთოდს გადაეცემა String persistenceUnitName პარამეტრი, რომელიც persistence.xml-ში გაწერილი გვაქვს <persistence-unit name="sampleDB"> ტეგის ატრიბუტად. ძირითადი ოპერაციების ჩასატარებლად კი გვჭირდება EntityManager ობიექტი, რომელსაც მივიღებთ EntityManagerFactory-ის createEntityManager() მეთოდით.

App კლასში ობიექტის ცხრილში ჩასაწერად (Insert) გამოიყენება EntityManager-ის persist მეთოდი. მიაქციეთ ყურადღება, რომ იგი ჩასმულია ტრანზაქციულ რეჟიმში: getTransaction().begin() და getTransaction().commit(). აღნიშნული მეთოდი ობიექტს ინახავს ცხრილში და იმავე წარმატებით შეგვიძლია გამოვიყენოთ Update-სთვისაც.

ცხრილიდან ჩანაწერების წასაშლელად გამოყენებულია EntityManager-ის remove მეთოდი.

ცხრილიდან ჩანაწრების წასაკითხად კი ვიყენებთ JPQL-ს (EJB QL), რომელიც ჰგავს კლასიკურ SQL-ს მაგრამ მასში ცხრილის სახელების ნაცვლად ვწერთ Entity კლასების და მისი ველების სახელებს და შესაბამისად ობიექტზე ორიენტირებულია.

ცხრილში ჩანაწერის შესაბამისი ობიექტის მონახვა პირველადი გასაღებით შესაძლებელია EntityManager-ის find მეთოდითაც (იხ. updateStudent მეთოდი). ობიექტის წაკითხვა ასევე შესაძლებელია EntityManager-ის getReference მეთოდითაც (იხ. deleteStudent მეთოდი), რომელიც მხოლოდ გვიბრუნებს ობიექტზე მიმთითებელს და მასში შევსებულია მხოლო პირველადი გასაღები ველები, ხოლო დანარჩენი ველები შეევსება მხოლოდ იმ შემთხვევაში, თუ მათ მივმართავთ (LAZY). ამით განსხვავდება ის find-სგან. ამ სხვაობის უკეთ წარმოსაჩენად, პროექტს დავამატოთ კლასები. მანამდე შევქმნათ ცხრილები:

**CREATE** **TABLE** Game **(**id BIGINT **NOT** **NULL,** name VARCHAR**(**255**),** **PRIMARY** **KEY** **(**id**));**

**CREATE** **TABLE** Player **(**id BIGINT **NOT** **NULL,** name VARCHAR**(**255**),** game\_id BIGINT**,** **PRIMARY** **KEY** **(**id**),** **FOREIGN** **KEY** **(**game\_id**)** **REFERENCES** Game**(**id**));**

package ge**.**ufc**.**jpa**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**Id**;**

@Entity

public class Game **{**

@Id

private Long id**;**

private String name**;**

public Game**()** **{**

**}**

public Game**(**Long id**,** String name**)** **{**

**this.**id **=** id**;**

**this.**name **=** name**;**

**}**

public Long getId**()** **{**

**return** id**;**

**}**

public void setId**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public String getName**()** **{**

**return** name**;**

**}**

public void setName**(**String name**)** **{**

**this.**name **=** name**;**

**}**

@Override

public String toString**()** **{**

**return** "Game [id=" **+** id **+** ", name=" **+** name **+** "]"**;**

**}**

**}**

---------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**ManyToOne**;**

@Entity

public class Player **{**

@Id

private Long id**;**

private String name**;**

@ManyToOne

private Game game**;**

public Player**()** **{**

**}**

public Player**(**Long id**,** String name**)** **{**

**this.**id **=** id**;**

**this.**name **=** name**;**

**}**

public Long getId**()** **{**

**return** id**;**

**}**

public void setId**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public String getName**()** **{**

**return** name**;**

**}**

public void setName**(**String name**)** **{**

**this.**name **=** name**;**

**}**

public Game getGame**()** **{**

**return** game**;**

**}**

public void setGame**(**Game game**)** **{**

**this.**game **=** game**;**

**}**

@Override

public String toString**()** **{**

**return** "Player [id=" **+** id **+** ", name=" **+** name **+** ", game=" **+** game **+** "]"**;**

**}**

**}**

-------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**persistence**.**EntityManager**;**

**import** javax**.**persistence**.**EntityManagerFactory**;**

**import** javax**.**persistence**.**Persistence**;**

public class App2 **{**

private static EntityManagerFactory entityManagerFactory**;**

private static EntityManager entityManager**;**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

entityManagerFactory **=** Persistence**.**createEntityManagerFactory**(**"sampleDB"**);**

entityManager **=** entityManagerFactory**.**createEntityManager**();**

entityManager**.**getTransaction**().**begin**();**

entityManager**.**persist**(new** Game**(**1L**,** "Game 1"**));**

entityManager**.**persist**(new** Game**(**2L**,** "Game 2"**));**

entityManager**.**persist**(new** Player**(**1L**,** "Player 1"**));**

entityManager**.**persist**(new** Player**(**2L**,** "Player 2"**));**

entityManager**.**persist**(new** Player**(**3L**,** "Player 3"**));**

entityManager**.**getTransaction**().**commit**();**

entityManager**.**clear**();**

entityManager**.**close**();**

**}**

public static void main**(**String**[]** args**)** **{**

**try** **{**

System**.**out**.**println**(**"\n\n"**);**

whenUsingFindMethodToUpdateGame\_thenExecutesSelectForGame**();** // select and update

whenUsingGetReferenceMethodToUpdateGame\_thenExecutesSelectForGame**();** // select and update

whenUsingFindMethodToDeletePlayer\_thenExecutesSelectForPlayer**();** // select and delete

whenUsingGetReferenceMethodToDeletePlayer\_thenExecutesSelectForPlayer**();** // select and delete

whenUsingFindMethodToUpdatePlayersGame\_thenExecutesSelectForGame**();** // select select and update

whenUsingGetReferenceMethodToUpdatePlayersGame\_thenDoesNotExecuteSelectForGame**();** // select and update

System**.**out**.**println**(**"\n\n"**);**

truncate**();**

**}** **finally** **{**

entityManagerFactory**.**close**();**

**}**

**}**

private static void whenUsingFindMethodToUpdateGame\_thenExecutesSelectForGame**()** **{**

runInTransaction**(()** **->** **{**

Game game1 **=** entityManager**.**find**(**Game**.**class**,** 1L**);**

game1**.**setName**(**"Game Updated 1"**);**

entityManager**.**persist**(**game1**);**

**});**

**}**

private static void whenUsingGetReferenceMethodToUpdateGame\_thenExecutesSelectForGame**()** **{**

runInTransaction**(()** **->** **{**

Game game1 **=** entityManager**.**getReference**(**Game**.**class**,** 1L**);**

game1**.**setName**(**"Game Updated 2"**);**

entityManager**.**persist**(**game1**);**

**});**

**}**

private static void whenUsingFindMethodToDeletePlayer\_thenExecutesSelectForPlayer**()** **{**

runInTransaction**(()** **->** **{**

Player player2 **=** entityManager**.**find**(**Player**.**class**,** 2L**);**

entityManager**.**remove**(**player2**);**

**});**

**}**

private static void whenUsingGetReferenceMethodToDeletePlayer\_thenExecutesSelectForPlayer**()** **{**

runInTransaction**(()** **->** **{**

Player player3 **=** entityManager**.**getReference**(**Player**.**class**,** 3L**);**

entityManager**.**remove**(**player3**);**

**});**

**}**

private static void truncate**()** **{**

runInTransaction**(()** **->** **{**

entityManager**.**createQuery**(**"delete from " **+** Player**.**class**.**getName**()).**executeUpdate**();**

entityManager**.**createQuery**(**"delete from " **+** Game**.**class**.**getName**()).**executeUpdate**();**

**});**

**}**

private static void whenUsingFindMethodToUpdatePlayersGame\_thenExecutesSelectForGame**()** **{**

runInTransaction**((()** **->** **{**

Game game1 **=** entityManager**.**find**(**Game**.**class**,** 1L**);**

Player player1 **=** entityManager**.**find**(**Player**.**class**,** 1L**);**

player1**.**setGame**(**game1**);**

entityManager**.**persist**(**player1**);**

**}));**

**}**

private static void whenUsingGetReferenceMethodToUpdatePlayersGame\_thenDoesNotExecuteSelectForGame**()** **{**

runInTransaction**(()** **->** **{**

Game game2 **=** entityManager**.**getReference**(**Game**.**class**,** 2L**);**

Player player1 **=** entityManager**.**find**(**Player**.**class**,** 1L**);**

player1**.**setGame**(**game2**);**

entityManager**.**persist**(**player1**);**

**});**

**}**

private static void runInTransaction**(**Runnable task**)** **{**

/\*

\* We create new persistence context for each test method to discard Hibernate

\* first level cache. So that we can see the behavior of getReference() method

\* in a non-cached environment.

\*/

entityManager **=** entityManagerFactory**.**createEntityManager**();**

entityManager**.**getTransaction**().**begin**();**

task**.**run**();**

entityManager**.**getTransaction**().**commit**();**

/\*

\* In any case, we use clear() and close() to make all the managed entities

\* detached and cleared. So, we can be sure we test always on a clear

\* persistence context.

\*/

entityManager**.**clear**();**

entityManager**.**close**();**

**}**

**}**

Player ცხრილი Game ცხრილთან დაკავშირებულია ბევრი ერთთან კავშირით. ანუ ერთ თამაშში შესაძლოა მონაწილეობდეს ბევრი მოთამაშე, მაგრამ ერთი მოთამაშე ყოველთვის ერთ თამაშში მონაწილეობს. კავშირებზე დაწვრილებით მოგვიანებით ვისაუბრებთ, ახლა კი ეს კავშირი ამ მაგალითში გამოგვადგება find და getReference მეთოდებში სხვაობის დასადგენად.

ჩავრთოთ show\_sql და გავუშვათ აპლიკაცია. მივიღებთ ასეთ შედეგს:

whenUsingFindMethodToUpdateGame\_thenExecutesSelectForGame - მეთოდში გაეშვება select და update.

whenUsingGetReferenceMethodToUpdateGame\_thenExecutesSelectForGame - მეთოდში გაეშვება select და update.

whenUsingFindMethodToDeletePlayer\_thenExecutesSelectForPlayer - მეთოდში გაეშვება select და delete.

whenUsingGetReferenceMethodToDeletePlayer\_thenExecutesSelectForPlayer - მეთოდში გაეშვება select და delete.

whenUsingFindMethodToUpdatePlayersGame\_thenExecutesSelectForGame - მეთოდში გაეშვება select, select და update, რადგან როგორც Game-ზე, ისე Player-ზე ვიყენებთ find-ს.

whenUsingGetReferenceMethodToUpdatePlayersGame\_thenDoesNotExecuteSelectForGame - რაც შეეხება ამ მეთოდს, Player-ს ვკითხულობთ find-ით (გაეშვება select ცხრილზე Player), ხოლო მისი ბაზაში შესანახად გვჭირდება დავუ-set-ოთ თავისი Game. Player ცხრილში თამაშზე მისაბმელად შენახულია მხოლოდ game\_id ველი ამიტომ ცხრილში ჩასაწერად მას მხოლოდ ეს ველი სჭირდება, ხოლო იგი გადაეცემა Game game2 = entityManager.getReference(Game.class, 2L); მეთოდის მეორე არგუმენტად (2) და ამიტომ JPA აღარ გაუშვებს Game ცხრილზე select-ს და შედეგად მივიღებთ ერთი select-ით ნაკლებს. ძირთადად ასეთ შემთხვევებაშია სხვაობა getReference და find მეთოდებს შორის.

შემდეგ მაგალითში ვნახოთ, როგორ უნდა მოვიქცეთ, თუ ცხრილის პირველადი გასაღები ერთზე მეტი ველისგან შედგება, ანუ უნიკალურია ერთდროულად რამდენიმე ველი - კომპოზიტური პირველადი გასაღები ადევს. ამ პრობლემის გადაწყვეტის ორი გზა არსებობს და ორივეს ვნახავთ. პირველ რიგში შევქმნათ შესაბამისი ცხრილები, სადაც ორი ველი ერთად არის პირველადი გასაღები.

**CREATE** **TABLE** Account **(**accountNumber VARCHAR**(**255**),** accountType VARCHAR**(**255**),** description VARCHAR**(**255**),** **PRIMARY** **KEY** **(**accountNumber**,** accountType**));**

**CREATE** **TABLE** Book **(**title VARCHAR**(**255**),** **language** VARCHAR**(**255**),** description VARCHAR**(**255**),** **PRIMARY** **KEY** **(**title**,** **language));**

პროექტს ჯერ დავამატოთ Account-ის შესაბამისი კლასები:

package ge**.**ufc**.**jpa**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**IdClass**;**

**import** javax**.**persistence**.**NamedQuery**;**

@Entity

@IdClass**(**AccountId**.**class**)**

@NamedQuery**(**name **=** "Account.findAccountByAccountId"**,** query **=** "from Account a where a.accountNumber = :accNum and a.accountType = :accType"**)**

public class Account **{**

@Id

private String accountNumber**;**

@Id

private String accountType**;**

private String description**;**

public String getAccountNumber**()** **{**

**return** accountNumber**;**

**}**

public void setAccountNumber**(**String accountNumber**)** **{**

**this.**accountNumber **=** accountNumber**;**

**}**

public String getAccountType**()** **{**

**return** accountType**;**

**}**

public void setAccountType**(**String accountType**)** **{**

**this.**accountType **=** accountType**;**

**}**

public String getDescription**()** **{**

**return** description**;**

**}**

public void setDescription**(**String description**)** **{**

**this.**description **=** description**;**

**}**

@Override

public String toString**()** **{**

**return** "Account [accountNumber=" **+** accountNumber **+** ", accountType=" **+** accountType **+** ", description="

**+** description **+** "]"**;**

**}**

**}**

----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** java**.**io**.**Serializable**;**

public class AccountId **implements** Serializable **{**

private static final long serialVersionUID **=** 1L**;**

private String accountNumber**;**

private String accountType**;**

public AccountId**()** **{**

**}**

public AccountId**(**String accountNumber**,** String accountType**)** **{**

**this.**accountNumber **=** accountNumber**;**

**this.**accountType **=** accountType**;**

**}**

@Override

public int hashCode**()** **{**

final int prime **=** 31**;**

int result **=** 1**;**

result **=** prime **\*** result **+** **((**accountNumber **==** **null)** **?** 0 **:** accountNumber**.**hashCode**());**

result **=** prime **\*** result **+** **((**accountType **==** **null)** **?** 0 **:** accountType**.**hashCode**());**

**return** result**;**

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(this** **==** obj**)**

**return** **true;**

**if** **(**obj **==** **null)**

**return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())**

**return** **false;**

AccountId other **=** **(**AccountId**)** obj**;**

**if** **(**accountNumber **==** **null)** **{**

**if** **(**other**.**accountNumber **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**accountNumber**.**equals**(**other**.**accountNumber**))**

**return** **false;**

**if** **(**accountType **==** **null)** **{**

**if** **(**other**.**accountType **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**accountType**.**equals**(**other**.**accountType**))**

**return** **false;**

**return** **true;**

**}**

**}**

აქ გამოვიყენეთ @IdClass(AccountId.class) ანოტაცია, ხოლო accountNumber/accountType პირველადი გასაღების ველები შევინარჩუნეთ Account კლასში და ორივეზე დავაწერეთ @Id ანოტაცია. რაც შეეხება AccountId კლასს, აქ ცალკე ასევე უნდა გვქონდეს accountNumber/accountType პირველადი გასაღების ველები და ეს კლასი აუცილებლად უნდა იყოს სერიალიზებადი.

ახლა ვნახოთ მეორე გზა:

package ge**.**ufc**.**jpa**;**

**import** javax**.**persistence**.**EmbeddedId**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**NamedQuery**;**

@Entity

@NamedQuery**(**name **=** "Book.findBookByBookId"**,** query **=** "from Book b where b.bookId.title = ?1 and b.bookId.language = ?2"**)**

public class Book **{**

@EmbeddedId

private BookId bookId**;**

private String description**;**

public Book**()** **{**

**}**

public Book**(**BookId bookId**)** **{**

**this.**bookId **=** bookId**;**

**}**

public BookId getBookId**()** **{**

**return** bookId**;**

**}**

public String getDescription**()** **{**

**return** description**;**

**}**

public void setDescription**(**String description**)** **{**

**this.**description **=** description**;**

**}**

@Override

public String toString**()** **{**

**return** "Book [bookId=" **+** bookId **+** ", description=" **+** description **+** "]"**;**

**}**

**}**

-------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** java**.**io**.**Serializable**;**

**import** javax**.**persistence**.**Embeddable**;**

@Embeddable

public class BookId **implements** Serializable **{**

private static final long serialVersionUID **=** 1L**;**

private String title**;**

private String language**;**

public BookId**()** **{**

**}**

public BookId**(**String title**,** String language**)** **{**

**this.**title **=** title**;**

**this.**language **=** language**;**

**}**

public String getTitle**()** **{**

**return** title**;**

**}**

public String getLanguage**()** **{**

**return** language**;**

**}**

@Override

public int hashCode**()** **{**

final int prime **=** 31**;**

int result **=** 1**;**

result **=** prime **\*** result **+** **((**language **==** **null)** **?** 0 **:** language**.**hashCode**());**

result **=** prime **\*** result **+** **((**title **==** **null)** **?** 0 **:** title**.**hashCode**());**

**return** result**;**

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(this** **==** obj**)**

**return** **true;**

**if** **(**obj **==** **null)**

**return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())**

**return** **false;**

BookId other **=** **(**BookId**)** obj**;**

**if** **(**language **==** **null)** **{**

**if** **(**other**.**language **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**language**.**equals**(**other**.**language**))**

**return** **false;**

**if** **(**title **==** **null)** **{**

**if** **(**other**.**title **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**title**.**equals**(**other**.**title**))**

**return** **false;**

**return** **true;**

**}**

@Override

public String toString**()** **{**

**return** "BookId [title=" **+** title **+** ", language=" **+** language **+** "]"**;**

**}**

**}**

აქაც დაგვჭირდა დამხარე კლასი BookId, რომელშიც შევინახეთ პირველადი გასაღების ველები title/language. ამ შემთხვევაში ეს კლასი მოვნიშნეთ @Embeddable ანოტაციით და აქაც მოთხოვნაა, რომ ეს კლასი იყოს სერიალიზებადი. რაც შეეხება Book კლასს, აქ პირველადი გასაღების ველად უკვე გვაქვს BookId კლასის ველი მონიშნული @EmbeddedId ანოტაციით. მთავარი სხვაობა ისაა, რომ ამჯერად Book კლასში პირდაპირ აღარ გვიწერია პირველადი გასაღები ველები და BookId ობიექტი გვაქვს ველად. ეს სხვაობას გამოიწევს შემდგომში დაწერილ JPQL-ებში. მაგალითად, Account-ის შემთხვევაში, თუ შეგვიძლია დავწეროთ Account.accountNumber და Account.accountType, Book-ის შემთხვევაში უნდა დავწეროთ Book.bookId.title და Book.bookId.language

ამის საილუსატრაციოდ, პროექტს დავამატოთ კიდევ ერთი სატესტო კლასი:

package ge**.**ufc**.**jpa**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**persistence**.**EntityManager**;**

**import** javax**.**persistence**.**EntityManagerFactory**;**

**import** javax**.**persistence**.**Persistence**;**

**import** javax**.**persistence**.**Query**;**

**import** javax**.**persistence**.**TypedQuery**;**

public class App3 **{**

private static final String SAVINGS\_ACCOUNT **=** "Savings"**;**

private static final String ACCOUNT\_NUMBER **=** "JXSDF324234"**;**

private static final String ENGLISH **=** "English"**;**

private static final String WAR\_AND\_PEACE **=** "War and Peace"**;**

private static EntityManagerFactory emf**;**

private static EntityManager em**;**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

emf **=** Persistence**.**createEntityManagerFactory**(**"sampleDB"**);**

em **=** emf**.**createEntityManager**();**

**}**

public static void main**(**String**[]** args**)** **{**

**try** **{**

persistAccountWithCompositeKeyThenRetrieveDetails**();**

persistBookWithCompositeKeyThenRetrieveDetails**();**

**}** **finally** **{**

truncate**();**

**if** **(**em **!=** **null)** **{**

em**.**close**();**

**}**

**if** **(**emf **!=** **null)** **{**

emf**.**close**();**

**}**

**}**

**}**

private static void persistAccountWithCompositeKeyThenRetrieveDetails**()** **{**

Account savingsAccount **=** createAccount**();**

persist**(**savingsAccount**);**

clearThePersistenceContext**();**

Account account **=** findAccountByAccountId**();**

System**.**out**.**println**(**account**);**

account **=** findAccountByAccountIdTypedQuery**();**

System**.**out**.**println**(**account**);**

account **=** findAccountByAccountIdNamedQuery**();**

System**.**out**.**println**(**account**);**

account **=** findAccountByAccountIdNativeQuery**();**

System**.**out**.**println**(**account**);**

**}**

private static void persistBookWithCompositeKeyThenRetrieveDetails**()** **{**

Book warAndPeace **=** createBook**();**

persist**(**warAndPeace**);**

clearThePersistenceContext**();**

Book book **=** findBookByBookId**();**

System**.**out**.**println**(**book**);**

book **=** findBookByBookIdTypedQuery**();**

System**.**out**.**println**(**book**);**

book **=** findBookByBookIdNamedQuery**();**

System**.**out**.**println**(**book**);**

book **=** findBookByBookIdNativeQuery**();**

System**.**out**.**println**(**book**);**

**}**

private static Account createAccount**()** **{**

Account savingsAccount **=** **new** Account**();**

savingsAccount**.**setAccountNumber**(**ACCOUNT\_NUMBER**);**

savingsAccount**.**setAccountType**(**SAVINGS\_ACCOUNT**);**

savingsAccount**.**setDescription**(**"Savings account"**);**

**return** savingsAccount**;**

**}**

private static Book createBook**()** **{**

BookId bookId **=** **new** BookId**(**WAR\_AND\_PEACE**,** ENGLISH**);**

Book warAndPeace **=** **new** Book**(**bookId**);**

warAndPeace**.**setDescription**(**"Novel and Historical Fiction"**);**

**return** warAndPeace**;**

**}**

private static Account findAccountByAccountId**()** **{**

**return** em**.**find**(**Account**.**class**,** **new** AccountId**(**ACCOUNT\_NUMBER**,** SAVINGS\_ACCOUNT**));**

**}**

private static Account findAccountByAccountIdTypedQuery**()** **{**

TypedQuery**<**Account**>** query **=** em

**.**createQuery**(**"from Account a where a.accountNumber = ?1 and a.accountType = :accType"**,** Account**.**class**);**

query**.**setParameter**(**1**,** ACCOUNT\_NUMBER**);**

query**.**setParameter**(**"accType"**,** SAVINGS\_ACCOUNT**);**

**return** query**.**getSingleResult**();**

**}**

private static Account findAccountByAccountIdNamedQuery**()** **{**

TypedQuery**<**Account**>** query **=** em**.**createNamedQuery**(**"Account.findAccountByAccountId"**,** Account**.**class**);**

query**.**setParameter**(**"accNum"**,** ACCOUNT\_NUMBER**);**

query**.**setParameter**(**"accType"**,** SAVINGS\_ACCOUNT**);**

**return** query**.**getSingleResult**();**

**}**

private static Account findAccountByAccountIdNativeQuery**()** **{**

Query query **=** em**.**createNativeQuery**(**"SELECT \* FROM Account WHERE accountNumber=?1 AND accountType=?2"**,**

Account**.**class**);**

query**.**setParameter**(**1**,** ACCOUNT\_NUMBER**);**

query**.**setParameter**(**2**,** SAVINGS\_ACCOUNT**);**

**return** **(**Account**)** query**.**getSingleResult**();**

**}**

private static Book findBookByBookId**()** **{**

**return** em**.**find**(**Book**.**class**,** **new** BookId**(**WAR\_AND\_PEACE**,** ENGLISH**));**

**}**

private static Book findBookByBookIdTypedQuery**()** **{**

TypedQuery**<**Book**>** query **=** em

**.**createQuery**(**"from Book b where b.bookId.title = ?1 and b.bookId.language = :language"**,** Book**.**class**);**

query**.**setParameter**(**1**,** WAR\_AND\_PEACE**);**

query**.**setParameter**(**"language"**,** ENGLISH**);**

**return** query**.**getSingleResult**();**

**}**

private static Book findBookByBookIdNamedQuery**()** **{**

TypedQuery**<**Book**>** query **=** em**.**createNamedQuery**(**"Book.findBookByBookId"**,** Book**.**class**);**

query**.**setParameter**(**1**,** WAR\_AND\_PEACE**);**

query**.**setParameter**(**2**,** ENGLISH**);**

**return** query**.**getSingleResult**();**

**}**

private static Book findBookByBookIdNativeQuery**()** **{**

Query query **=** em**.**createNativeQuery**(**"SELECT \* FROM Book b WHERE b.title = :title and b.language = :language"**,** Book**.**class**);**

query**.**setParameter**(**"title"**,** WAR\_AND\_PEACE**);**

query**.**setParameter**(**"language"**,** ENGLISH**);**

**return** **(**Book**)** query**.**getSingleResult**();**

**}**

private static void persist**(**Account account**)** **{**

em**.**getTransaction**().**begin**();**

em**.**persist**(**account**);**

em**.**getTransaction**().**commit**();**

**}**

private static void persist**(**Book book**)** **{**

em**.**getTransaction**().**begin**();**

em**.**persist**(**book**);**

em**.**getTransaction**().**commit**();**

**}**

private static void clearThePersistenceContext**()** **{**

em**.**clear**();**

**}**

private static void truncate**()** **{**

em**.**getTransaction**().**begin**();**

em**.**createQuery**(**"delete from " **+** Account**.**class**.**getName**()).**executeUpdate**();**

em**.**createQuery**(**"delete from " **+** Book**.**class**.**getName**()).**executeUpdate**();**

em**.**getTransaction**().**commit**();**

**}**

**}**

findAccountByAccountIdTypedQuery-ში გამოყენებულია from Account a where a.accountNumber = ?1 and a.accountType = :accType ხოლო from Book b where b.bookId.title = ?1 and b.bookId.language = :language JPQL ბრძანებები. მიაქციეთ ყუარდღება პარამეტრებსაც, ზოგგან გამოყენებულია პოზიციებით გადანომრვა (?1), ზოგგან - კი სახელებით (:accType, :language).

მაგალითში ასევე გამოვიყენთ NamedQuery მეთოდებში findBookByBookIdNamedQuery/findAccountByAccountIdNamedQuery. იგივე JPQL-ები დავაწერეთ კლასებს @NamedQuery ანოტაციებით:

@NamedQuery(name = "Book.findBookByBookId", query = "from Book b where b.bookId.title = ?1 and b.bookId.language = ?2")

@NamedQuery(name = "Account.findAccountByAccountId", query = "from Account a where a.accountNumber = :accNum and a.accountType = :accType")

@NamedQuery-ში დარქმეული name სახელით უკვე მარტივად შეგვიძლია ბრძანებაზე წვდომა:

TypedQuery<Account> query = em.createNamedQuery("Account.findAccountByAccountId", Account.class);

TypedQuery<Book> query = em.createNamedQuery("Book.findBookByBookId", Book.class);

მაგალითში ასევე გამოვიყენეთ NativeQuery, სადაც უკვე იწერება კლასიკური sql ბრძანებები, რომლებსაც ვწერთ უშუალოდ ბაზასთან ინტერაქციაში (მაგალითად, ij-ში), მისი სინტაქსი მორგებულია მთლიანად ბაზაზე და კლასების და კლასის ველების სახელების ნაცვლად შეიცავს ცხრილებს და მათს ველებს (როგორც ვწერდით JDBC-ში).

Query query = em.createNativeQuery("SELECT \* FROM Account WHERE accountNumber=?1 AND accountType=?2", Account.class);

Query query = em.createNativeQuery("SELECT \* FROM Book b WHERE b.title = :title and b.language = :language", Book.class);

# Hibernate

შემდეგი მაგალითი უკვე ავაწყოთ უშუალოდ Hibernate-ის კლასებით, რომლებიც Java EE-ს JPA-ში არ შედის და Hibernate-ის დამატებული რეალიზაციაა. შესაბამისად კოდი იმუშავებს მხოლოდ Hibernate jpa პროვაიდერში.

აქ მნიშვნელოვანია SessionFactory ინტერფეისი, რომელიც jpa-ს javax.persistence.EntityManagerFactory მემკვიდრეა და Session ინტერფეისი, რომელიც მემკვიდრეა jpa-ს javax.persistence.EntityManager-ის, რომელიც ჩვენთვის უკვე ცნობილია.

სანამ პროექტს დავაგენერირებთ, ბაზაში შევქმნათ სატესტო ცხრილი და შევავსოთ მონაცემებით:

**CREATE** **TABLE** cars**(**id INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY,** name VARCHAR**(**255**),** price INT**);**

**INSERT** **INTO** cars**(**name**,** price**)** **VALUES** **(**'Audi'**,** 52642**),** **(**'Mercedes'**,** 57127**),** **(**'Skoda'**,** 9000**),** **(**'Volvo'**,** 29000**),** **(**'Bentley'**,** 350000**),** **(**'Citroen'**,** 21000**),** **(**'Hummer'**,** 41400**),** **(**'Volkswagen'**,** 21600**);**

დავაგენერიროთ ახალი პროექტი: mvn archetype:generate -DgroupId=ge.ufc.jpa -DartifactId=jpa-hibernate-2 -Dversion=1.0 -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false

pom.xml

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<project xmlns=**"http://maven.apache.org/POM/4.0.0"** xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"** xsi:schemaLocation=**"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"**>

<modelVersion>**4.0.0**</modelVersion>

<groupId>**ge.ufc.pooling**</groupId>

<artifactId>**jpa-hibernate-2**</artifactId>

<version>**1.0**</version>

<name>**jpa-hibernate-2**</name>

<url>**http://www.ufc.ge**</url>

<properties>

<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>

<maven.compiler.release>**11**</maven.compiler.release>

<derby.version>**10.15.2.0**</derby.version>

<hibernate.version>**5.6.12.Final**</hibernate.version>

</properties>

<dependencies>

<dependency>

<groupId>**org.hibernate**</groupId>

<artifactId>**hibernate-core**</artifactId>

<version>**${hibernate.version}**</version>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbyclient**</artifactId>

<version>**${derby.version}**</version>

<scope>**runtime**</scope>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbytools**</artifactId>

<version>**${derby.version}**</version>

<scope>**runtime**</scope>

</dependency>

</dependencies>

<build>

<finalName>**jpaHibernate2**</finalName>

<plugins>

<plugin>

<artifactId>**maven-compiler-plugin**</artifactId>

<version>**3.8.1**</version>

</plugin>

<plugin>

<groupId>**org.apache.maven.plugins**</groupId>

<artifactId>**maven-eclipse-plugin**</artifactId>

<version>**2.10**</version>

<configuration>

<projectNameTemplate>

**[artifactId]-[version]**

</projectNameTemplate>

<wtpmanifest>**true**</wtpmanifest>

<wtpapplicationxml>**true**</wtpapplicationxml>

<wtpversion>**2.0**</wtpversion>

<manifest>

**${basedir}/src/main/resources/META-INF/MANIFEST.MF**

</manifest>

</configuration>

</plugin>

<plugin>

<artifactId>**maven-assembly-plugin**</artifactId>

<version>**3.4.2**</version>

<executions>

<execution>

<phase>**package**</phase>

<goals>

<goal>**single**</goal>

</goals>

<configuration>

<descriptorRefs>

<descriptorRef>**jar-with-dependencies**</descriptorRef>

</descriptorRefs>

<appendAssemblyId>**false**</appendAssemblyId>

<archive>

<manifest>

<mainClass>**ge.ufc.jpa.App**</mainClass>

</manifest>

</archive>

</configuration>

</execution>

</executions>

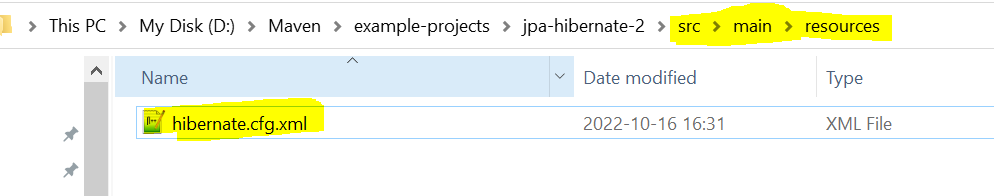
</plugin>

</plugins>

</build>

</project>

ამჯერად კონფიგურაციულ ფაილს ჰქვია hibernate.cfg.xml და განთავსებულია პროექტის src\main\resources დირექტორიაში. მისი ფორმატი განსხვავდება persistence.xml-სგან, მაგრამ შინაარსობრივად იგივე მახასითებლები და მნიშვნელობებია გაწერილი.



<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=**"hibernate.connection.driver\_class"**>**org.apache.derby.jdbc.ClientDriver**</property>

<property name=**"hibernate.connection.url"**>**jdbc:derby://localhost:1527/sample**</property>

<property name=**"hibernate.connection.username"**>**APP**</property>

<property name=**"hibernate.connection.password"**>**123**</property>

<property name=**"hibernate.dialect"**>**org.hibernate.dialect.DerbyTenSevenDialect**</property>

<property name=**"hibernate.show\_sql"**>**true**</property>

<mapping class=**"ge.ufc.jpa.bean.Car"**></mapping>

<mapping class=**"ge.ufc.jpa.bean.Item"**></mapping>

</session-factory>

</hibernate-configuration>

აქ უბრალოდ დავამატეთ ბაზის username/password, რომელიც როგორც ვიცით, დერბის შემთხვევაში არაა აუცილებელი. ასევე აქ უნდა გავწეროთ Entity კლასები mapping-ში.

პროექტს დავამატოთ კლასები:

package ge**.**ufc**.**jpa**.**bean**;**

**import** java**.**util**.**Objects**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**GeneratedValue**;**

**import** javax**.**persistence**.**GenerationType**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "cars"**)**

public class Car **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

private int id**;**

private String name**;**

private Integer price**;**

public int getId**()** **{**

**return** id**;**

**}**

public void setId**(**int id**)** **{**

**this.**id **=** id**;**

**}**

public String getName**()** **{**

**return** name**;**

**}**

public void setName**(**String name**)** **{**

**this.**name **=** name**;**

**}**

public Integer getPrice**()** **{**

**return** price**;’**

**}**

public void setPrice**(**Integer price**)** **{**

**this.**price **=** price**;**

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(this** **==** obj**)**

**return** **true;**

**if** **(**obj **==** **null** **||** getClass**()** **!=** obj**.**getClass**())**

**return** **false;**

Car car **=** **(**Car**)** obj**;**

**return** Objects**.**equals**(this.**id**,** car**.**id**)** **&&** Objects**.**equals**(this.**name**,** car**.**name**)** **&&** Objects**.**equals**(this.**price**,** car**.**price**);**

**}**

@Override

public int hashCode**()** **{**

**return** Objects**.**hash**(**id**,** name**,** price**);**

**}**

@Override

public String toString**()** **{**

final StringBuilder sb **=** **new** StringBuilder**(**"Car{"**);**

sb**.**append**(**"Id="**).**append**(**id**);**

sb**.**append**(**", Name='"**).**append**(**name**).**append**(**'\''**);**

sb**.**append**(**", Price="**).**append**(**price**);**

sb**.**append**(**'}'**);**

**return** sb**.**toString**();**

**}**

**}**

--------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** org**.**hibernate**.**boot**.**Metadata**;**

**import** org**.**hibernate**.**boot**.**MetadataSources**;**

**import** org**.**hibernate**.**boot**.**registry**.**StandardServiceRegistryBuilder**;**

**import** org**.**hibernate**.**service**.**ServiceRegistry**;**

public class HibernateUtil **{**

private static SessionFactory sessionFactory**;**

private HibernateUtil**()** **{**

**}**

public static SessionFactory getSessionFactory**()** **{**

**if** **(**sessionFactory **==** **null)** **{**

sessionFactory **=** buildSessionFactory**();**

**}**

**return** sessionFactory**;**

**}**

private static SessionFactory buildSessionFactory**()** **{**

**try** **{**

ServiceRegistry registry **=** **new** StandardServiceRegistryBuilder**().**configure**(**"hibernate.cfg.xml"**).**build**();**

MetadataSources metadataSources **=** **new** MetadataSources**(**registry**);**

Metadata metadata **=** metadataSources**.**getMetadataBuilder**().**build**();**

**return** metadata**.**getSessionFactoryBuilder**().**build**();**

**}** **catch** **(**Exception ex**)** **{**

**throw** **new** ExceptionInInitializerError**(**ex**);**

**}**

**}**

public static void shutdown**()** **{**

**if** **(**sessionFactory **!=** **null)** **{**

sessionFactory**.**close**();**

**}**

**}**

**}**

--------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** org**.**hibernate**.**Session**;**

**import** ge**.**ufc**.**jpa**.**bean**.**Car**;**

public class App **{**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

**}**

public static void main**(**String**[]** args**)** **{**

System**.**out**.**println**(**getCarById**(**2**));**

System**.**out**.**println**(**getCars**());**

Car car **=** **new** Car**();**

car**.**setName**(**"Toyota"**);**

car**.**setPrice**(**34500**);**

addCar**(**car**);**

System**.**out**.**println**(**getCarById**(**car**.**getId**()));**

car**.**setName**(**"Toyota RAV 4"**);**

car**.**setPrice**(**37000**);**

updateCar**(**car**);**

System**.**out**.**println**(**getCarById**(**car**.**getId**()));**

deleteCar**(**car**);**

System**.**out**.**println**(**getCarById**(**car**.**getId**()));**

**}**

private static Car getCarById**(**int id**)** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

**return** session**.**get**(**Car**.**class**,** id**);**

**}**

**}**

private static List**<**Car**>** getCars**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

**return** session**.**createQuery**(**"from Car"**,** Car**.**class**).**list**();**

**}**

**}**

private static void addCar**(**Car car**)** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

session**.**beginTransaction**();**

session**.**save**(**car**);**

session**.**getTransaction**().**commit**();**

**}**

**}**

private static void updateCar**(**Car car**)** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

session**.**beginTransaction**();**

session**.**update**(**car**);**

session**.**getTransaction**().**commit**();**

**}**

**}**

private static void deleteCar**(**Car car**)** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

session**.**beginTransaction**();**

session**.**delete**(**car**);**

session**.**getTransaction**().**commit**();**

**}**

**}**

**}**

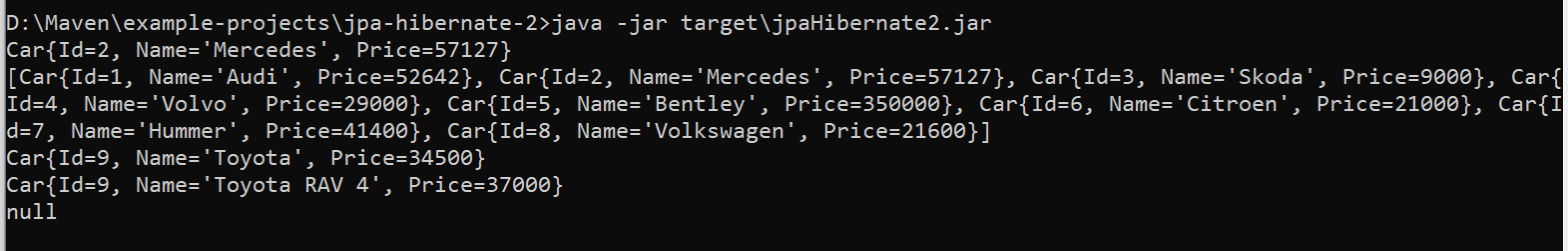
Car კლასი აქაც jpa-ს ანოტაციებითაა გაფორმებული, რომლებიც უკვე ნაცნობია ჩვენთვის.

პირველ რიგში ყურადღება უნდა მივაქციოთ HibernateUtil singleton კლასს, რომელიც გვეხმარება org.hibernate.SessionFactory ობიექტის მიღებაში და კონფიგრუაციას კითხულობს ჩვენ მიერ შექმნილი hibernate.cfg.xml ფაილიდან.

რაც შეეხება მთავარ App კლასს, როგორც ხედავთ, აქ EntityManager-ის ნაცვლად გვაქვს org.hibernate.Session კლასი. ობიექტის ცხრილში ჩასაწერად (Insert) persist-ის ნაცვლად გვაქვს save მეთოდი, გასაახლებლად persist-ის ნაცვლად - update მეთოდი, წასაშლელად remove-ს ნაცვლად - delete მეთოდი, ხოლო მოსაძებნად find-ის ნაცვლად get მეთოდი. ტრანზაქციის დასაწყებად getTransaction().begin()-ის ნაცვლად გვაქვს პირდაპირ beginTransaction, ხოლო კომიტზე აქაც გვჭირდება getTransaction().commit(). SQL ბრძანებების გასაშვებად აქ გვაქვს HQL, რომელიც თითქმის JPQL-ის იდენტურია და ისიც ობიექტზე ორიენტირებულია, რა თქმა უნდა.

mvn clean package გავბილდოთ პროექტი და გავუშვათ: java -jar target\jpaHibernate2.jar

უნდა მივიღოთ ასეთი შედეგი:



შემდეგ მაგალითში უნდა ვისწავლოთ JPA-ს CriteriaAPI, რომელიც გამოიყენება SQL ბრძანებების JPQL, HQL-ის გარეშე ასაწყობად, ოპერირება ხდება ობიექტებზე და კლასებზე და მიუხედავად იმისა, რომ კოდი უფრო იზრდება და რთულად კითხვადი ხდება, შეცდომების ალბათობა უფრო მცირდება, რადგან შეხება გვაქვს არა სტრიქონთან, არამედ ობიექტებთან და კლასებთან.

პროექტის App კლასი გადავწეროთ CriteriaAPI-ით App2 კლასში:

package ge**.**ufc**.**jpa**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**persistence**.**criteria**.**CriteriaBuilder**;**

**import** javax**.**persistence**.**criteria**.**CriteriaDelete**;**

**import** javax**.**persistence**.**criteria**.**CriteriaQuery**;**

**import** javax**.**persistence**.**criteria**.**CriteriaUpdate**;**

**import** javax**.**persistence**.**criteria**.**Predicate**;**

**import** javax**.**persistence**.**criteria**.**Root**;**

**import** org**.**hibernate**.**Session**;**

**import** ge**.**ufc**.**jpa**.**bean**.**Car**;**

public class App2 **{**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

**}**

public static void main**(**String**[]** args**)** **{**

System**.**out**.**println**(**getCarById**(**2**));**

System**.**out**.**println**(**getCars**());**

Car car **=** **new** Car**();**

car**.**setName**(**"Toyota"**);**

car**.**setPrice**(**34500**);**

addCar**(**car**);**

System**.**out**.**println**(**getCarById**(**car**.**getId**()));**

car**.**setName**(**"Toyota RAV 4"**);**

car**.**setPrice**(**37000**);**

updateCar**(**car**);**

System**.**out**.**println**(**getCarById**(**car**.**getId**()));**

deleteCar**(**car**);**

System**.**out**.**println**(**getCarById**(**car**.**getId**()));**

**}**

private static Car getCarById**(**int id**)** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Car**>** criteria **=** builder**.**createQuery**(**Car**.**class**);**

Root**<**Car**>** root **=** criteria**.**from**(**Car**.**class**);**

Predicate predicate **=** builder**.**equal**(**root**.**get**(**"id"**),** id**);**

criteria**.**where**(**predicate**);**

**return** session**.**createQuery**(**criteria**).**uniqueResult**();**

**}**

**}**

private static List**<**Car**>** getCars**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Car**>** criteria **=** builder**.**createQuery**(**Car**.**class**);**

criteria**.**select**(**criteria**.**from**(**Car**.**class**));**

**return** session**.**createQuery**(**criteria**).**list**();**

**}**

**}**

private static void addCar**(**Car car**)** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

session**.**beginTransaction**();**

session**.**save**(**car**);**

session**.**getTransaction**().**commit**();**

**}**

**}**

private static void updateCar**(**Car car**)** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaUpdate**<**Car**>** criteriaUpdate **=** builder**.**createCriteriaUpdate**(**Car**.**class**);**

Root**<**Car**>** root **=** criteriaUpdate**.**from**(**Car**.**class**);**

criteriaUpdate**.**set**(**"name"**,** car**.**getName**()).**set**(**"price"**,** car**.**getPrice**());**

criteriaUpdate**.**where**(**builder**.**equal**(**root**.**get**(**"id"**),** car**.**getId**()));**

session**.**beginTransaction**();**

session**.**createQuery**(**criteriaUpdate**).**executeUpdate**();**

session**.**getTransaction**().**commit**();**

**}**

**}**

private static void deleteCar**(**Car car**)** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaDelete**<**Car**>** criteriaDelete **=** builder**.**createCriteriaDelete**(**Car**.**class**);**

Root**<**Car**>** root **=** criteriaDelete**.**from**(**Car**.**class**);**

criteriaDelete**.**where**(**builder**.**equal**(**root**.**get**(**"id"**),** car**.**getId**()));**

session**.**beginTransaction**();**

session**.**createQuery**(**criteriaDelete**).**executeUpdate**();**

session**.**getTransaction**().**commit**();**

**}**

**}**

**}**

როგორც ხედავთ, მონაცემების წამოსაღებად გვჭირდება CriteriaQuery, გასაახლებლად - CriteriaUpdate, ხოლო წასაშლელად - CriteriaDelete კლასები. ცხრილის მისათითებლად ვიყენებთ from (არაა აუცილებელი) მეთოდს, ხოლო ფილტრაციის პირობის განსასაზღვრად where მეთოდს. მონაცემების წამოსაღებად ვიყენებთ list მეთოდს, თუ ველოდებით ერთზე მეტ ობიექტს, ხოლო თუ ერთ ობიექტს ველოდებით, ვიყენებთ uniqueResult მეთოდს. რაც შეეხება update და delete-ს, მათ გასაშვებად გვაქვს executeUpdate (JDBC-ის ანალოგიურად) მეთოდი, რომელიც აბრუნებს განახლებული/წაშლილი ჩანაწერების რაოდენობას (მთელ რიცხვს). თუ დაგვიბრუნა 0, ნიშნავს, რომ ცხრილში ცვლილება არ მომხდარა, ძალიან დიდი ალბათობით, ფილტრაციის პირობის ვერ შესრულების გამო.Update-ის შემთხვევაში ველებზე ახალი მნიშვნელობების მისანიჭებლად გამოიყენება set მეთოდი.

Criteria API-ს კიდევ უფრო კარგად გასაცნობად, გავაკეთოთ მორიგი მაგალითი. შევქმნათ გასატესტად საჭირო ცხრილი და შევავსოთ მონაცემებით:

**CREATE** **TABLE** ITEM **(**ITEM\_ID INTEGER **NOT** **NULL** **PRIMARY** **KEY,** ITEM\_DESC VARCHAR**(**100**),** ITEM\_PRICE INTEGER **NOT** **NULL,** ITEM\_NAME VARCHAR**(**100**)** **NOT** **NULL);**

**INSERT** **INTO** ITEM **(**ITEM\_ID**,** ITEM\_NAME**,** ITEM\_DESC**,** ITEM\_PRICE**)** **VALUES**

**(**1**,** 'item One'**,** 'test 1'**,** 35.12**),**

**(**2**,** 'Pogo stick'**,** 'Pogo stick'**,** 466.12**),**

**(**3**,** 'Raft'**,** 'Raft'**,** 345.12**),**

**(**4**,** 'Skate Board'**,** 'Skating'**,** 135.71**),**

**(**5**,** 'Umbrella'**,** 'Umbrella for Rain'**,** 619.25**),**

**(**6**,** 'Glue'**,** 'Glue for home'**,** 432.73**),**

**(**7**,** 'Paint'**,** 'Paint for Room'**,** 1311.40**),**

**(**8**,** 'Red paint'**,** 'Red paint for room'**,** 1135.71**),**

**(**9**,** 'Household Chairs'**,** 'Chairs for house'**,** 25.71**),**

**(**10**,** 'Office Chairs'**,** 'Chairs for office'**,** 395.98**),**

**(**11**,** 'Outdoor Chairs'**,** 'Chairs for outdoor activities'**,** 1234.36**);**

პროექტს დავამატოთ კლასები:

package ge**.**ufc**.**jpa**.**bean**;**

**import** java**.**io**.**Serializable**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "ITEM"**)**

public class Item **implements** Serializable **{**

private static final long serialVersionUID **=** 1L**;**

@Id

@Column**(**name **=** "ITEM\_ID"**)**

private Integer itemId**;**

@Column**(**name **=** "ITEM\_NAME"**)**

private String itemName**;**

@Column**(**name **=** "ITEM\_DESC"**)**

private String itemDescription**;**

@Column**(**name **=** "ITEM\_PRICE"**)**

private Integer itemPrice**;**

// constructors

public Item**()** **{**

**}**

public Item**(**final Integer itemId**,** final String itemName**,** final String itemDescription**)** **{**

**super();**

**this.**itemId **=** itemId**;**

**this.**itemName **=** itemName**;**

**this.**itemDescription **=** itemDescription**;**

**}**

@Override

public int hashCode**()** **{**

final int prime **=** 31**;**

int result **=** 1**;**

result **=** prime **\*** result **+** **((**itemId **==** **null)** **?** 0 **:** itemId**.**hashCode**());**

**return** result**;**

**}**

@Override

public boolean equals**(**final Object obj**)** **{**

**if** **(this** **==** obj**)**

**return** **true;**

**if** **(**obj **==** **null)**

**return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())**

**return** **false;**

final Item other **=** **(**Item**)** obj**;**

**if** **(**itemId **==** **null)** **{**

**if** **(**other**.**itemId **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**itemId**.**equals**(**other**.**itemId**))**

**return** **false;**

**return** **true;**

**}**

public Integer getItemId**()** **{**

**return** itemId**;**

**}**

public void setItemId**(**final Integer itemId**)** **{**

**this.**itemId **=** itemId**;**

**}**

public String getItemName**()** **{**

**return** itemName**;**

**}**

public void setItemName**(**final String itemName**)** **{**

**this.**itemName **=** itemName**;**

**}**

public String getItemDescription**()** **{**

**return** itemDescription**;**

**}**

public Integer getItemPrice**()** **{**

**return** itemPrice**;**

**}**

public void setItemPrice**(**final Integer itemPrice**)** **{**

**this.**itemPrice **=** itemPrice**;**

**}**

public void setItemDescription**(**final String itemDescription**)** **{**

**this.**itemDescription **=** itemDescription**;**

**}**

**}**

-------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** java**.**util**.**List**;**

**import** javax**.**persistence**.**criteria**.**CriteriaBuilder**;**

**import** javax**.**persistence**.**criteria**.**CriteriaQuery**;**

**import** javax**.**persistence**.**criteria**.**Predicate**;**

**import** javax**.**persistence**.**criteria**.**Root**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**query**.**Query**;**

**import** ge**.**ufc**.**jpa**.**bean**.**Item**;**

public class ItemApp **{**

public boolean checkIfCriteriaTimeLower**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**root**).**where**(**builder**.**like**(**root**.**get**(**"itemName"**),** "%item One%"**));**

long startTimeCriteria **=** System**.**nanoTime**();**

session**.**createQuery**(**criteria**).**list**();**

long endTimeCriteria **=** System**.**nanoTime**();**

long durationCriteria **=** endTimeCriteria **-** startTimeCriteria**;**

System**.**out**.**println**(**"durationCriteria: " **+** durationCriteria**);**

long startTimeHQL **=** System**.**nanoTime**();**

session**.**createQuery**(**"FROM Item where itemName like '%item One%'"**).**list**();**

long endTimeHQL **=** System**.**nanoTime**();**

long durationHQL **=** endTimeHQL **-** startTimeHQL**;**

System**.**out**.**println**(**"durationHQL: " **+** durationHQL**);**

**return** durationCriteria **>** durationHQL**;**

**}**

**}**

// To get items having price more than 1000

public String**[]** greaterThanCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**root**).**where**(**builder**.**gt**(**root**.**get**(**"itemPrice"**),** 1000**));** //gt - greaterThan

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

// To get items having price less than 1000

public String**[]** lessThanCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**root**).**where**(**builder**.**lt**(**root**.**get**(**"itemPrice"**),** 1000**));** //lt - lessThan

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

// To get items whose Name start with Chair

public String**[]** likeCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**root**).**where**(**builder**.**like**(**root**.**get**(**"itemName"**),** "Chair%"**));**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

//To get records having itemPrice in between 100 and 200

public String**[]** betweenCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**root**).**where**(**builder**.**between**(**root**.**get**(**"itemPrice"**),** 100**,** 200**));**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

//To get records having itemName in 'Skate Board', 'Paint' and 'Glue'

public String**[]** inCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**root**).**where**(**root**.**get**(**"itemName"**).**in**(**"Skate Board"**,** "Paint"**,** "Glue"**));**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

// To check if the given property is null

public String**[]** nullCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**root**).**where**(**builder**.**isNull**(**root**.**get**(**"itemDescription"**)));**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

// To check if the given property is not null

public String**[]** notNullCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**root**).**where**(**builder**.**isNotNull**(**root**.**get**(**"itemDescription"**)));**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

// Adding more than one expression in one cr

public String**[]** twoCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

Predicate**[]** predicates **=** **new** Predicate**[**2**];**

predicates**[**0**]** **=** builder**.**isNull**(**root**.**get**(**"itemDescription"**));**

predicates**[**1**]** **=** builder**.**like**(**root**.**get**(**"itemName"**),** "chair%"**);**

criteria**.**select**(**root**).**where**(**predicates**);**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

//To get items matching with the above defined conditions joined with Logical AND

public String**[]** andLogicalCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

Predicate greaterThanPrice **=** builder**.**gt**(**root**.**get**(**"itemPrice"**),** 1000**);**

Predicate chairItems **=** builder**.**like**(**root**.**get**(**"itemName"**),** "Chair%"**);**

criteria**.**select**(**root**).**where**(**builder**.**and**(**greaterThanPrice**,** chairItems**));**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

// To get items matching with the above defined conditions joined with Logical OR

public String**[]** orLogicalCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

Predicate greaterThanPrice **=** builder**.**gt**(**root**.**get**(**"itemPrice"**),** 1000**);**

Predicate chairItems **=** builder**.**like**(**root**.**get**(**"itemName"**),** "Chair%"**);**

criteria**.**select**(**root**).**where**(**builder**.**or**(**greaterThanPrice**,** chairItems**));**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

// Sorting example

public String**[]** sortingCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**root**);**

criteria**.**orderBy**(**builder**.**asc**(**root**.**get**(**"itemName"**)),** builder**.**desc**(**root**.**get**(**"itemPrice"**)));**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

// Set projections Row Count

public Long**[]** projectionRowCount**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Long**>** criteria **=** builder**.**createQuery**(**Long**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**builder**.**count**(**root**));**

Query**<**Long**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Long**>** itemProjected **=** query**.**getResultList**();**

Long projectedRowCount**[]** **=** **new** Long**[**itemProjected**.**size**()];**

**for** **(**int i **=** 0**;** i **<** itemProjected**.**size**();** i**++)** **{**

projectedRowCount**[**i**]** **=** itemProjected**.**get**(**i**);**

**}**

**return** projectedRowCount**;**

**}**

**}**

// Set projections average of itemPrice

public Double**[]** projectionAverage**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Double**>** criteria **=** builder**.**createQuery**(**Double**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

criteria**.**select**(**builder**.**avg**(**root**.**get**(**"itemPrice"**)));**

Query**<**Double**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Double**>** avgItemPriceList **=** query**.**getResultList**();**

Double avgItemPrice**[]** **=** **new** Double**[**avgItemPriceList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** avgItemPriceList**.**size**();** i**++)** **{**

avgItemPrice**[**i**]** **=** avgItemPriceList**.**get**(**i**);**

**}**

**return** avgItemPrice**;**

**}**

**}**

public String**[]** combineTwoOrUsingAndCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

Predicate glueItem **=** builder**.**equal**(**root**.**get**(**"itemName"**),** "Glue"**);**

Predicate paintItem **=** builder**.**equal**(**root**.**get**(**"itemName"**),** "Paint"**);**

Predicate glueOrPaintItem **=** builder**.**or**(**glueItem**,** paintItem**);**

Predicate price1Item **=** builder**.**lt**(**root**.**get**(**"itemPrice"**),** 1200**);**

Predicate price2Item **=** builder**.**gt**(**root**.**get**(**"itemPrice"**),** 1300**);**

Predicate priceItem **=** builder**.**or**(**price1Item**,** price2Item**);**

Predicate finalPredicate **=** builder**.**and**(**glueOrPaintItem**,** priceItem**);**

criteria**.**select**(**root**).**where**(**finalPredicate**);**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

public String**[]** combineTwoAndUsingOrCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Item**>** criteria **=** builder**.**createQuery**(**Item**.**class**);**

Root**<**Item**>** root **=** criteria**.**from**(**Item**.**class**);**

Predicate paintItem1 **=** builder**.**like**(**root**.**get**(**"itemName"**),** "%Paint%"**);**

Predicate paintItem2 **=** builder**.**like**(**root**.**get**(**"itemName"**),** "%paint%"**);**

Predicate paintItem **=** builder**.**and**(**paintItem1**,** paintItem2**);**

Predicate price1Item **=** builder**.**gt**(**root**.**get**(**"itemPrice"**),** 1100**);**

Predicate price2Item **=** builder**.**lt**(**root**.**get**(**"itemPrice"**),** 1400**);**

Predicate priceItem **=** builder**.**and**(**price1Item**,** price2Item**);**

Predicate finalPredicate **=** builder**.**or**(**paintItem**,** priceItem**);**

criteria**.**select**(**root**).**where**(**finalPredicate**);**

Query**<**Item**>** query **=** session**.**createQuery**(**criteria**);**

List**<**Item**>** greaterThanItemsList **=** query**.**getResultList**();**

String greaterThanItems**[]** **=** **new** String**[**greaterThanItemsList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** greaterThanItemsList**.**size**();** i**++)** **{**

greaterThanItems**[**i**]** **=** greaterThanItemsList**.**get**(**i**).**getItemName**();**

**}**

**return** greaterThanItems**;**

**}**

**}**

**}**

-------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** java**.**util**.**Arrays**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** org**.**hibernate**.**Session**;**

**import** ge**.**ufc**.**jpa**.**bean**.**Item**;**

public class ItemAppTester **{**

private static final ItemApp app**;**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

app **=** **new** ItemApp**();**

**}**

public static void main**(**String**[]** args**)** **{**

System**.**out**.**println**(**app**.**checkIfCriteriaTimeLower**());** //true

testGreaterThanCriteria**();**

testLessThanCriteria**();**

testLikeCriteria**();**

testBetweenCriteria**();**

testInCriteria**();**

testNullCriteria**();**

testNotNullCriteria**();**

testTwoCriteria**();**

testAndLogicalCriteria**();**

testOrLogicalCriteria**();**

testSortingCriteria**();**

testRowCountProjection**();**

testAverageProjection**();**

testCombineTwoOrUsingAndCriteria**();**

testCombineTwoAndUsingOrCriteria**();**

**}**

private static void testGreaterThanCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemPrice > 1000"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**greaterThanCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testLessThanCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemPrice < 1000"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**lessThanCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testLikeCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemName like 'Chair%'"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**likeCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testBetweenCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemPrice between 100 and 200"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**betweenCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testInCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemName in ('Skate Board', 'Paint', 'Glue')"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**inCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testNullCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemDescription is null"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**nullCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testNotNullCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemDescription is not null"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**notNullCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testTwoCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemDescription is null and itemName like 'chair%'"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**twoCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testAndLogicalCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemPrice>1000 and itemName like 'Chair%'"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**andLogicalCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testOrLogicalCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where itemPrice>1000 or itemName like 'Chair%'"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**orLogicalCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testSortingCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item order by itemName asc, itemPrice desc"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**sortingCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testRowCountProjection**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Long**>** expectedList **=** session**.**createQuery**(**"Select count(\*) from Item"**).**list**();**

Long **[]** expectedItems **=** **new** Long**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**);**

**}**

Long **[]** returnedItems **=** app**.**projectionRowCount**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testAverageProjection**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Double**>** expectedList **=** session**.**createQuery**(**"Select avg(itemPrice) from Item item"**).**list**();**

Double **[]** expectedItems **=** **new** Double**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**);**

**}**

Double **[]** returnedItems **=** app**.**projectionAverage**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testCombineTwoOrUsingAndCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where (itemName = 'Glue' or itemName = 'Paint') and (itemPrice < 1200 or itemPrice > 1300)"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**combineTwoOrUsingAndCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

private static void testCombineTwoAndUsingOrCriteria**()** **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**Item**>** expectedList **=** session**.**createQuery**(**"From Item where (itemName like '%Paint%' and itemName like '%paint%') or (itemPrice > 1100 and itemPrice < 1400)"**).**list**();**

String **[]** expectedItems **=** **new** String**[**expectedList**.**size**()];**

**for** **(**int i **=** 0**;** i **<** expectedList**.**size**();** i**++)** **{**

expectedItems**[**i**]** **=** expectedList**.**get**(**i**).**getItemName**();**

**}**

String **[]** returnedItems **=** app**.**combineTwoAndUsingOrCriteria**();**

System**.**out**.**println**(**Arrays**.**equals**(**expectedItems**,** returnedItems**));** //true

**}**

**}**

**}**

ItemApp კლასში მეთოდები დაწერილია CriteriaBuilder/CriteriaQuery/Predicate კლასებით და ყველა მეთოდი აღწერს თავის თავს. ვიყენებთ სხვადასხვა ფილტრაციებს: მეტობა/ნაკლებობაზე შედარებას რიცხვით ტიპებზე, ტოლობაზე შედარებას, არის თუ არ არის მნიშვნელობა NULL, სტრიქონში like-ით ძებნას, between-ს შუალედში ძებნისთვის, in-ს კონკრეტულ მნიშვნელობათა სიმრავლეში ძებნისთვის, ლოგიკურ „და“ და „ან“-ს და ასევე პირობებში მათს კომბინაციებს და ა.შ. ვიყენებთ ასევე აგრეგატულ ფუნქციებს: რაოდენობის (COUNT) და საშუალო არითმეტიკულის გამოთვლა (AVG). გვაქვს ასევე სორიტრების მაგალითიც.

ItemAppTester კლასში კი, მეთოდების გასატესტად შესაბამისი ბრძანებები დაწერილი გვაქვს HQL-ით და თითოეულ მეთოდზე ვამოწმებთ, რომ CriteriaAPI-ით მიღებული შემდეგი იდენტურია HQL-ით მიღებული შედეგისა.

# Hibernate: save, persist, update, merge, saveOrUpdate

ამ ლექციაში ვიმსჯელებთ, რა განსხვავებაა Session ინტერფეისის save, persist, update, merge და saveOrUpdate მეთოდებში.

## Session as a Persistence Context Implementation

Session ინტერფეისის აქვს რამდენიმე მეთოდი, რომლებიც დაკავშირებული არიან მონაცემების ბაზაში შენახვასთან: save, persist, update, merge და saveOrUpdate. მათში არსებული სხვაობების გასაგებად, პირველ რიგში უნდა გვესმოდეს Session-ის დანიშნულება, როგორც persistence context-ის და entity ობიექტების მდგომარეობები სესიაში. ასევე უნდა ვიცოდეთ Hibernate-ის დეველოპმენტის ისტორია, რომელმაც API-ს მეთოდების ნაწილობრივი დუბლირება გამოიწვია.

## Managing Entity Instances

ობიექტურ-რელაციური კავშირების გარდა, ერთ-ერთი მთავარი პრობლემა, რასაც Hibernate წყვეტს, არის Runtime-ში entity-ების მართვა. აქ შემოდის persistence context ცნება, როგორც პრობლემის გადაწყვეტა. იგი წარმოადგებს კონტეინერს ან პირველი დონის ქეშს ყველა ობიექტისა, რომლებსაც ვტვირთავთ და ვინახავთ ბაზაში სესიის მიმდინარეობის დროს. სესია არის ლოგიკური ტრანზაქცია აპლიკაციის ბიზნეს ლოგიკით განსაზღვრული საზღვრებით. როდესაც ჩვენ ვმუშაობთ მონაცემთა ბაზასთან persistence context-ით და ყველა ჩვენი Entity-ს ობიექტი მიბმულია ამ კონტექსტზე, ჩვენ აუცილებლად უნდა გვქონდეს ერთი Entity-ს ობიექტის ეგზემპლარი დაკავშირებული ბაზაში ერთ ჩანაწერთან, რომელსაც ვემუშავებით სესიაში.

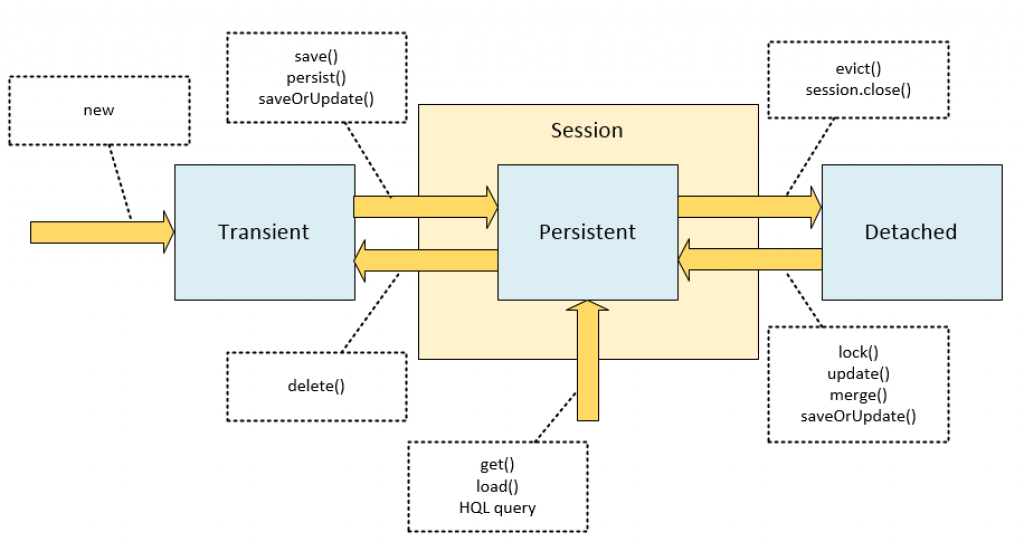
Hibernate-ში persistence context წარმოდგენილია org.hibernate.Session ეგზემპლარით, ხოლო JPA-ში javax.persistence.EntityManager-ით. როდესაც ვიყენებთ Hibernate-ს JPA პროვაიდერად და ოპერაციებს ვახდენთ EntityManager ინტერფეისზე, ამ ინტერფეისის მემკვიდრეა Session ინტერფეისი, რომელიც გვათავაზობს დამატებით შესაძლებლობებს. ამიტომ ზოგჯერ უმჯობესია, პირდაპირ Session ინტერფეისს ვემუშაოთ.

## States of Entity Instances

ნებისმიერ Entity-ს ობიექტს Session-ის persistence context-ში აქვს ჩამოთვლილი სამი მდგომარეობიდან ერთი:

1. **transient** (გამჭვირვალე, ნეიტრალური) - ასეთი ეგზემპლარები არ არიან და არასდროს იყვნენ სესიაზე მიბმული. მათ არ გააჩნიათ შესაბამისი ჩანაწერები ბაზის ცხრილში. ისინი წარმოადგენენ უბრალოდ ახალ ობიექტებს, რომლებიც შევქმენით ბაზაში შესანახად.
2. **persistent** (მუდმივი, შენახული) - ეგზემპლარები უნიკალურად მიბმული არიან სესიაზე. სესიის ბაზაში გადატანის (გადაწერის, flush) შემდეგ ობიექტს გარანტირებულად ექნება შესაბამისი ჩანაწერი ცხრილში.
3. **detached** (კონტექსტიდან ამოღებული, ჩახსნილი) - ეგზემპლარი მანამდე (ადრე) მიბმული იყო სესიაზე (შესაბამის ცხრილში), მაგრამ ახლა აღარაა - ჩაიხსნა სესიის კონტექსტიდან. ობიექტი გადადის ამ მდგომარეობაში, თუ მას ჩავხსნით სესიიდან evict მეთოდით, ან დავხურავთ (clear) ან დავაცარიელებთ სესიას (close) ან ეგზემპლარს მოვაშორებთ სერიალიზაცია/დესერიალიზაციის პროცესისგან.

მდგომარეობები აღწერილია შემდეგ დიაგრამაზე. ასევე დიაგრამზე აღწერილია, რომელ მეთოდებს რომელი მდგომარეობიდან რომელ მდგომარეობაში გადაჰყავთ ობიექტები.



როდესაც entity-ის ეგზემპლარი არის persistent მდგომარეობაში, მის ყველა ველზე ჩატარებული ცვლილება აისახება ბაზის ჩანაწერებში, როდესაც სესია დაიხურება (flush, close). persistent ეგზემპლარი არის online, ხოლო detached - offline და მისი ცვლილებები არ აისახება ბაზაში.

ეს ნიშნავს შემდეგს, რომ როდესაც ჩვენ ვცვლით persistent ობიექტის ველებს, ამ ცვლილების ბაზის ცხრილის შესაბამის ჩანაწერში ასასახად აღარ გვჭირდება save ან update-ის გამოძახება, საკმარისია დავაკომიტოთ ტრანზაქცია ან დავხუროთ ან დავაცარიელოთ სესია (commit, flush or close).

## Conformity to JPA Specification

Hibernate წარმოადგენს Java ORM-ის ერთ-ერთ ყველაზე წარმატებულ რეალიზაციას, თუმცა მან თავად მნიშვნელოვანი ცვლილებები დაუმატა Java persistence API (JPA)-ს: ზოგი საკმაოდ მნიშვნელოვანი და ზოგიც დახვეწილი.

EntityManager-ის მემკვიდრე Session ინტერფეისში დაემატა რიგი მეთოდები. ეს მეთოდები ემსახურება იმავე მიზანს, როგორც ორიგინალური მეთოდები, მაგრამ შეესაბამება სპეციფიკაციას და შესაბამისად, აქვთ გარკვეული განსხვავებები.

# Differences Between the Operations

პირველ რიგში აუცილებელია ვიცოდეთ, რომ ეს მეთოდები (persist, save, update, merge, saveOrUpdate) მაშინვე მყისიერად არ აისახება SQL UPDATE ან INSERT ბრძანებებში. ცვლილება ძალაში შედის ტრანზაქციის დაკომიტების ან სესიის დახურვის შემდეგ. ამ მეთოდების მთავარი მიზანი კი სესიაში ობიექტების ერთი მდგომარეობიდან მეორეში გადაყავანაა.

მაგალითებისთვის გამოვიყენოთ მარტივი კლასი Person:

@Entity

public class Person **{**

@Id

@GeneratedValue

private Long id**;**

private String name**;**

// ... getters and setters

**}**

## Persist

persist გამოიყენება ობიექტის persistence context ჩასასმელად - მდგომარეობა იცვლება transient-დან persistence-ზე. მეთოდი შედის JPA-ს სპეციფიკაციაში.

როგორც წესი, მას ვიყენებთ ჩანაწერის ბაზაში დასამატებლად:

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**persist**(**person**);**

persist მეთოდის გამოძახების შემდეგ ობიექტის მდგომარეობა იცვლება transient-დან persistence-ზე, მაგრამ ცვლილება არ არის ჯერ ბაზაში ასახული. INSERT ბრძანება გაეშვება მხოლოდ კომიტის ან სესიის დაცარიელბა/დახურვის შემდეგ.

persist მეთოდის დასაბრუნებელი ტიპია void. მისი გამოძახების შემდეგ persist მეთოდი მიუთითებს აქტუალურ persistence ობიექტზე.

* თუკი ობიექტის ეგზემპლარი უკვე არის persistence მდგომარეობაში (persistence კონტექსტში), მაშინ persist მეთოდის გამოძახება არაფერს ცვლის.
* თუკი ობიექტის ეგზემპლარი არის detached მდგომარეობაში, persist მეთოდის გამოძახება ან სესიის დაკომიტება/დახურვა გამოიწვევს Exception-ს.

კომიტის ან სესიის დაცარიელბა/დახურვის შემდეგ, თუ ცხრილს აქვს ავტომაურად გენერირებადი ველი, დაგენერირებული მნიშვნელობა აისახება persistence ობიექტზეც.

ქვემოთ მოყვანილ მაგალითში ვახდენთ ობიექტის persist-ს, შემდეგ ვაკეთებთ მისი სესიიდან ჩახსნას evict მეთოდით და შემდეგ ისევ ვცდილობთ persist-ის გამოძახებას, რაც არ იმუშავებს და გამოიწვევს Exception-ს:

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**persist**(**person**);**

session**.**evict**(**person**);**

session**.**persist**(**person**);** // PersistenceException!

## Save

save წარმოადგენს Hibernate-ს სპეციფიურ მეთოდს და იგი არაა აღწერილი JPA სპეციფიკაციაში.

მისი მიზანი ანალოგიურია persist მეთოდისა, მაგრამ რეალიზაციაში გვაქვს მნიშვნელოვანი სხვაობები. დოკუმენტაციის მიხედვით იგი ობიექტის persist-ს ახდენს პირველ რიგში იდენტიფიკატორის გენერაციით. იგი აბრუნებს ამ იდენტიფიკატორის სერიალიზებულ მნიშვნელობას:

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

Long id **=** **(**Long**)** session**.**save**(**person**);**

უკვე persisted ობიექტის save-ის გავლენა იდენტურია persist მეთოდისა, ანუ რეალურად არაფერს არ ცვლის. სხვაობაა detached ეგზემპლარზე save-ის გამოძახებაში. persist-სგან განსხვავებით აქ არ ხდება Exception-ის გამოსროლა და ობიექტი ახალ ჩანაწერად მიდის ბაზაში:

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

Long id1 **=** **(**Long**)** session**.**save**(**person**);**

session**.**evict**(**person**);**

Long id2 **=** **(**Long**)** session**.**save**(**person**);**

ზემოთ მოყვანილ მაგალითში id2 განსხვავდება id1-სგან. ანუ save მეთოდმა detached ეგზემპლარი ისევ persistent მდგომარეობაში გადაიყვანა და ახალი id2 დააგენერირა, რაც გამოიწევს ბაზაში ჩანაწერის დუბლირებას. ანუ persist-სგან განსხვავებით აქ Exception არ მიგვიღია.

## Merge

merge მეთოდის მთავარი დანიშნულებაა persistent ეგზემპლარის განახლება ახალი ველებით detached ეგზემპლარიდან. მეთოდი შედის JPA-ს სპეციფიკაციაში.

მაგალითად, გვაქვს RESTful ინტერფეისი მეთოდით JSON-serialized ობიექტის ამოსაღებად (მოსაძებნად) მისი id-ით და მეორე მეთოდი, რომელიც გამომძახებლისგან იღებს ამ ობიექტის განახლებულ ვერსიას. სერიალიზაცია/დესერიალიზაციის პროცესში ობიექტი აღმოჩნდება detached მდგომარეობაში.

ობიექტის დესერიალიზაციის შემდეგ გვჭირდება persistent ობიექტი, რომელსაც ველების მნიშვნელობებს განვუახლებთ detached ეგზემპლარიდან. ამას აკეთებს merge მეთოდი:

* პოულობს entity ეგზემპლარს გადაცემული ობიექტის id-ით (ან არსებული ეგზემპლარი ბრუნდება persistence context-დან ან ახალი ეგზემპლარი იტვირთება ბაზიდან).
* გადაცემული ობიექტის ველებს აკოპირებს ამ ეგზემპლარში.
* აბრუნებს ახალ განახლებულ ეგზემპლარს.

შემდეგ მაგალითში, ჩვენ კონტექსტიდან ჩავხსნით ობიექტს, შევუცვლით name ველს და დავმერჯავთ ჩახსნილ ობიექტს:

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**save**(**person**);**

session**.**evict**(**person**);**

person**.**setName**(**"Mary"**);**

Person mergedPerson **=** **(**Person**)** session**.**merge**(**person**);**

მიაქციეთ ყურადღება, რომ merge მეთოდი აბრუნებს ობიექტს. ეს გახლავთ persistence context-ში არსებული და განახლებული mergedPerson ობიექტი, რომელიც განსხვავდება არგუმენტში გადაცემული person ობიექტისგან. ანუ ისინი ორი სხვდასხვა ობიექტებია და როგორც წესი, person ობიექტი აღარ დაგვჭირდება.

JSR-220 სპეციფიკაციის მიხედვით:

* თუ გადაცემული ობიექტი არის detached მდგომარეობაში, ის გადაეწერება არსებულ persistent ობიექტს.
* თუ გადაცემული ობიექტი არის transient მდგომარეობაში, ის გადაეწერება ახლად შექმნილ persistent ობიექტს.
* თუ გადაცემული ობიექტი არის persistent მდგომარეობაში, megre მეთოდის გამოძახება არაფერს ცვლის.

## Update

save-ის მსგავსად update არის Hibernate-ის მიერ დამატებული. მისი სემანტიკა განსხვავდება რამდენიმე ძირითადი პუნქტით:

* ის ოპერირებს გადაცემულ ობიექტზე და მისი დასაბრუნებელი ტიპი არის void. მეთოდს detached-დან persistent მდგომარეობაში გადაჰყავს ობიექტი.
* მეთოდი ისვრის Exception-ს, თუ არგუმენტად გადავეცით transient მდგომარეობაში მყოფი ობიექტი.

ქვემოთ მოყვანილ მაგალითში, ჩვენ ვინახავთ ობიექტს persistence context-ში, შემდეგ ჩავხსნით მას (evict) კონტექსტიდან, ვცვლით მის სახელს და ვიძახებთ update-ს. დააკვირდით, რომ update მეთოდი არის void, არ აბრუნებს განალხებულ ობიექტს და განახლებას უკეთებს არგუმენტში გადაცემულ ობიექტს. შესაბამისად, update მეთოდის საშუალებით ჩვენ ისევ ვაბრუნებთ ობიექტს სესიის კონტექსტში, რაც არ შეგვეძლო JPA სპეციფიკაციებით (merge მხოლოდ გადაწერას აკეთებდა, მაგრამ არგუმენტად გადაცემული ობიექტი არ ბრუნდებოდა სესიის კონტექსტში):

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**save**(**person**);**

session**.**evict**(**person**);**

person**.**setName**(**"Mary"**);**

session**.**update**(**person**);**

update მეთოდის გამოძახება transient ობიექტზე გამოიწვევს Exception-ს:

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**update**(**person**);** // PersistenceException!

## SaveOrUpdate

ეს მეთოდიც არის მხოლოდ Hibernate-ში და არც მას გააჩნია სტანდარტიზებული ანალოგი. update მეთოდის მსგავსად, ისიც შეგვიძლია ობიექტის context-ში დასაბრუნებლად გამოვიყენოთ.

მთავარია სხვაობა update-თან შედარებით არის ის, რომ იგი არ ისვრის Exception-ს, transient ობიექტი თუ გადავეცით და ამ ობიექტს ხდის persistence-ს. შემდეგი კოდი Person-ის ახლად შექმნილ ობიექტს გადაიყვანს persist მდგომარეობაში:

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**saveOrUpdate**(**person**);**

ეს მეთოდი შეგვიძლია განვიხილოთ, როგორც უნივერსალური ხელსაწყო ობიექტის persistent მდგომარეობაში გადასაყვანად, არ აქვს მნიშვნელობა ობიექტი transient არის თუ detached.

# What to Use?

თუ არ გვაქვს სპეციფიური მოთხოვნები, ჯობია გამოვიყენოთ persist და merge მეთოდები, რადგან ისინი სტანდარტიზეულია JPA სპეციფიკაციით. ისინი არიან პორტაბელური, თუ გადავწყვეტთ ახალი persistence პროვაიდერის გამოყენებას. თუმცა ზოგჯერ Hibernate-ის save, update და saveOrUpdate მეთოდები უფრო გამოსადეგარია.

# Conclusion

ამ ლექციაში განვიხილეთ Hibernate სესიის სხვადასხვა დანიშნულების მეთოდები persistent ობიექტების სამართავად. ვისწავლეთ, როგორ გარდაქმნიან ეს მეთოდები ობიექტების მდგომარეობებს და რატომ აქვს ზოგიერთ მეთოდს ფუნქციონალის დუბლირება გაკეთებული.

save, persist, update, merge, saveOrUpdate, evict მეთოდების გასატესტად ბაზას დავამატოთ ცხრილი: **CREATE** **TABLE** Person**(**id INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY,** name VARCHAR**(**255**));**

hibernate.cfg.xml კონფიგურაციის ფაილში ჩავამატოთ:

<mapping class=**"ge.ufc.jpa.bean.Person"**></mapping>

ხოლო პროექტს დავამატოთ კლასები:

package ge**.**ufc**.**jpa**.**bean**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**GeneratedValue**;**

**import** javax**.**persistence**.**GenerationType**;**

**import** javax**.**persistence**.**Id**;**

@Entity

public class Person **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

private Long id**;**

private String name**;**

public Long getId**()** **{**

**return** id**;**

**}**

public void setId**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public String getName**()** **{**

**return** name**;**

**}**

public void setName**(**String name**)** **{**

**this.**name **=** name**;**

**}**

@Override

public String toString**()** **{**

**return** "Person [id=" **+** id **+** ", name=" **+** name **+** "]"**;**

**}**

**}**

--------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** javax**.**persistence**.**PersistenceException**;**

**import** org**.**hibernate**.**HibernateException**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** ge**.**ufc**.**jpa**.**bean**.**Person**;**

public class SaveMethodsTest **{**

private static SessionFactory sessionFactory**;**

private Session session**;**

private boolean doNotCommit**;**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

sessionFactory **=** HibernateUtil**.**getSessionFactory**();**

**}**

**{**

doNotCommit **=** **false;**

session **=** sessionFactory**.**openSession**();**

session**.**beginTransaction**();**

**}**

public static void main**(**String**[]** args**)** **{**

SaveMethodsTest test **=** **null;**

**try** **{**

test **=** **new** SaveMethodsTest**();**

test**.**whenPersistTransient\_thenSavedToDatabaseOnCommit**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenPersistPersistent\_thenNothingHappens**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenPersistDetached\_thenThrowsException**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenMergeDetached\_thenEntityUpdatedFromDatabase**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenSaveTransient\_thenIdGeneratedImmediately**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenSavePersistent\_thenNothingHappens**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenSaveDetached\_thenNewInstancePersisted**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenMergeTransient\_thenNewEntitySavedToDatabase**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenMergePersistent\_thenReturnsSameObject**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenUpdateDetached\_thenEntityUpdatedFromDatabase**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenUpdateTransient\_thenThrowsException**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenUpdatePersistent\_thenNothingHappens**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenSaveOrUpdateDetached\_thenEntityUpdatedFromDatabase**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenSaveOrUpdateTransient\_thenSavedToDatabaseOnCommit**();**

test **=** **new** SaveMethodsTest**();**

test**.**whenSaveOrUpdatePersistent\_thenNothingHappens**();**

**}** **finally** **{**

**if** **(!**test**.**doNotCommit**)** **{**

test**.**session**.**getTransaction**().**commit**();**

**}**

test**.**session**.**close**();**

sessionFactory**.**close**();**

**}**

**}**

private void whenPersistTransient\_thenSavedToDatabaseOnCommit**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**persist**(**person**);**

session**.**getTransaction**().**commit**();**

session**.**close**();**

session **=** sessionFactory**.**openSession**();**

session**.**beginTransaction**();**

System**.**out**.**println**(**session**.**get**(**Person**.**class**,** person**.**getId**()));** // person not null

**}**

private void whenPersistPersistent\_thenNothingHappens**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**persist**(**person**);**

Long id1 **=** person**.**getId**();**

session**.**persist**(**person**);**

Long id2 **=** person**.**getId**();**

System**.**out**.**println**(**id1 **+** " - " **+** id2**);** // should be the same

**}**

private void whenPersistDetached\_thenThrowsException**()** **{**

// doNotCommit = true;

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**persist**(**person**);**

session**.**evict**(**person**);**

**try** **{**

session**.**persist**(**person**);**

**}** **catch** **(**PersistenceException e**)** **{**

System**.**out**.**println**(**e**.**getMessage**());** // detached entity passed to persist

**}**

**}**

private void whenMergeDetached\_thenEntityUpdatedFromDatabase**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**save**(**person**);**

session**.**flush**();**

session**.**evict**(**person**);**

person**.**setName**(**"Mary"**);**

Person mergedPerson **=** **(**Person**)** session**.**merge**(**person**);**

System**.**out**.**println**(**person **!=** mergedPerson**);** // should be true, person and mergedPerson are not the same

System**.**out**.**println**(**mergedPerson**.**getName**());** // Mary

**}**

private void whenSaveTransient\_thenIdGeneratedImmediately**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

System**.**out**.**println**(**person**.**getId**());** // Should be null

Long id **=** **(**Long**)** session**.**save**(**person**);**

System**.**out**.**println**(**person**.**getId**());** // Should be not null

session**.**getTransaction**().**commit**();**

session**.**close**();**

System**.**out**.**println**(**id **==** person**.**getId**());** // Should be the same

session **=** sessionFactory**.**openSession**();**

session**.**beginTransaction**();**

System**.**out**.**println**(**session**.**get**(**Person**.**class**,** person**.**getId**()));**

**}**

private void whenSavePersistent\_thenNothingHappens**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

Long id1 **=** **(**Long**)** session**.**save**(**person**);**

Long id2 **=** **(**Long**)** session**.**save**(**person**);**

System**.**out**.**println**(**id1 **+** " - " **+** id2**);** // Should be the same

**}**

private void whenSaveDetached\_thenNewInstancePersisted**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

Long id1 **=** **(**Long**)** session**.**save**(**person**);**

session**.**evict**(**person**);**

Long id2 **=** **(**Long**)** session**.**save**(**person**);**

System**.**out**.**println**(**id1 **+** " - " **+** id2**);** // Should not be the same: id2 = id1 + 1

**}**

private void whenMergeTransient\_thenNewEntitySavedToDatabase**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

Person mergedPerson **=** **(**Person**)** session**.**merge**(**person**);**

session**.**getTransaction**().**commit**();**

session**.**beginTransaction**();**

System**.**out**.**println**(**person**.**getId**());** //Should be null

System**.**out**.**println**(**mergedPerson**.**getId**());** //Should not be null

**}**

private void whenMergePersistent\_thenReturnsSameObject**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**save**(**person**);**

Person mergedPerson **=** **(**Person**)** session**.**merge**(**person**);**

System**.**out**.**println**(**person **==** mergedPerson**);** //Should be true

**}**

private void whenUpdateDetached\_thenEntityUpdatedFromDatabase**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**save**(**person**);**

session**.**evict**(**person**);**

person**.**setName**(**"Mary"**);**

session**.**update**(**person**);**

System**.**out**.**println**(**person**.**getName**());** //Mary

**}**

private void whenUpdateTransient\_thenThrowsException**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

**try** **{**

session**.**update**(**person**);**

**}** **catch** **(**HibernateException e**)** **{**

System**.**out**.**println**(**e**.**getMessage**());** //The given object has a null identifier

**}**

**}**

private void whenUpdatePersistent\_thenNothingHappens**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**save**(**person**);**

session**.**update**(**person**);**

**}**

private void whenSaveOrUpdateDetached\_thenEntityUpdatedFromDatabase**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**save**(**person**);**

session**.**evict**(**person**);**

person**.**setName**(**"Mary"**);**

session**.**saveOrUpdate**(**person**);**

System**.**out**.**println**(**person**.**getName**());** //Mary

**}**

private void whenSaveOrUpdateTransient\_thenSavedToDatabaseOnCommit**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**saveOrUpdate**(**person**);**

session**.**getTransaction**().**commit**();**

session**.**close**();**

session **=** sessionFactory**.**openSession**();**

session**.**beginTransaction**();**

System**.**out**.**println**(**session**.**get**(**Person**.**class**,** person**.**getId**()));**

**}**

private void whenSaveOrUpdatePersistent\_thenNothingHappens**()** **{**

Person person **=** **new** Person**();**

person**.**setName**(**"John"**);**

session**.**save**(**person**);**

session**.**saveOrUpdate**(**person**);**

**}**

**}**

# Entity Relationships

ამ ლექციაში განვიხილავთ ცხრილებს შორის არსებულ რელაციურ კავშირებს JPA-ში.

## One-to-One Relationship in JPA

ამ ნაწილში ვისწავლით, როგორ ავაწყოთ ერთი-ერთთან რელაციური კავშირი JPA-ში.

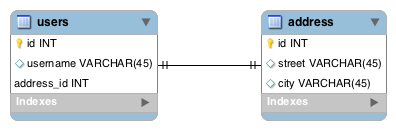
### Description

ვთქვათ, ვაწყობთ მომხმარებლების მართვის სისტემას და ბაზაში უნდა შევინახოთ მომხმარებლები და მათი მისამართები. ერთ მომხარებელს აქვს ყოველთვის ერთი მისამართი და ერთ მისამართზეც ყოველთვის ერთი მომხარებელია მიბმული.

შესაბამისად, გვექნება User და Address entity-ები დაკავშირებული ერთი-ერთთან კავშირით.

### Using a Foreign Key

დავაკვირდეთ ER (Entity relationship model) დიაგრამას, რომელიც ერთ-ერთთან კავშირს აღწერს გარე გასაღებით. users ცხრილის address\_id ველი წარმოადგენს address ცხრილის id ველის გარე გასაღებს:



### Implementing With a Foreign Key in JPA

ანოტაციებით გაფორმებული User კლასი გამოიყურება ასე:

@Entity

@Table**(**name **=** "users"**)**

public class User **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**AUTO**)**

@Column**(**name **=** "id"**)**

private Long id**;**

//...

@OneToOne**(**cascade **=** CascadeType**.**ALL**)**

@JoinColumn**(**name **=** "address\_id"**,** referencedColumnName **=** "id"**)**

private Address address**;**

// ... getters and setters

**}**

რელაციურად დაკავშირებულ Address ველზე დავაწერეთ ანოტაცია @OneToOne. ასევე გამოვიყენეთ @JoinColumn ანოტაცია users ცხრილში სვეტის სახელის განსასაზღვრად, რომელიც დაუკავშირდება address ცხრილში პირველად გასაღებს, რომელი სვეტის სახელიც განსაზღვრულია referencedColumnName ატრიბუტით.

მეორე Entity-ში კი აღარ გვჭირდება @JoinColumn ანოტაცია, რადგან იგი გამოიყენება მხოლოდ კავშირის მფლობელის მხარეს. მარტივად რომ ვთქვათ, ვისაც აქვს გარე გასაღების სვეტი, მას ეწერება @JoinColumn ანოტაცია.

@Entity

@Table**(**name **=** "address"**)**

public class Address **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**AUTO**)**

@Column**(**name **=** "id"**)**

private Long id**;**

//...

@OneToOne**(**mappedBy **=** "address"**)**

private User user**;**

//... getters and setters

**}**

ორმხრივი კავშირის დასამყარებლად, Address კლასშიც გამოვიყენეთ @OneToOne ანოტაცია mappedBy ატრიბუტით, რომლის მნიშვნელობაა User-კლასის address ველის სახელი. mappedBy რომელ კლასშიც წერია, ის კლასი არ არის კავშირის მფლობელი მხარე.

გასატესტად ბაზაში დავამატოთ შესაბამისი ცხრილები:

**CREATE** **TABLE** address**(**id INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY,** street VARCHAR**(**45**),** city VARCHAR**(**45**));**

**CREATE** **TABLE** users**(**id INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY,** username VARCHAR**(**45**),** address\_id INTEGER**,** **FOREIGN** **KEY(**address\_id**)** **REFERENCES** address**(**id**),** **Unique** **(**address\_id**));**

hibernate.cfg.xml კონფიგურაციის ფაილში ჩავამატოთ:

<mapping class=**"ge.ufc.jpa.one\_to\_one.User"**></mapping>

<mapping class=**"ge.ufc.jpa.one\_to\_one.Address"**></mapping>

ხოლო პროექტს დავამატოთ კლასები:

package ge**.**ufc**.**jpa**.**one\_to\_one**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**GeneratedValue**;**

**import** javax**.**persistence**.**GenerationType**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**OneToOne**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "address"**)**

public class Address **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

@Column**(**name **=** "id"**)**

private Long id**;**

@Column**(**name **=** "street"**)**

private String street**;**

@Column**(**name **=** "city"**)**

private String city**;**

@OneToOne**(**mappedBy **=** "address"**)**

private User user**;**

public Long getId**()** **{**

**return** id**;**

**}**

public void setId**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public String getStreet**()** **{**

**return** street**;**

**}**

public void setStreet**(**String street**)** **{**

**this.**street **=** street**;**

**}**

public String getCity**()** **{**

**return** city**;**

**}**

public void setCity**(**String city**)** **{**

**this.**city **=** city**;**

**}**

public User getUser**()** **{**

**return** user**;**

**}**

public void setUser**(**User user**)** **{**

**this.**user **=** user**;**

**}**

**}**

----------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**one\_to\_one**;**

**import** javax**.**persistence**.**CascadeType**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**GeneratedValue**;**

**import** javax**.**persistence**.**GenerationType**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**JoinColumn**;**

**import** javax**.**persistence**.**OneToOne**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "users"**)**

public class User **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

@Column**(**name **=** "id"**)**

private Long id**;**

@Column**(**name **=** "username"**)**

private String userName**;**

@OneToOne**(**cascade **=** CascadeType**.**ALL**)**

@JoinColumn**(**name **=** "address\_id"**,** referencedColumnName **=** "id"**)**

private Address address**;**

public Long getId**()** **{**

**return** id**;**

**}**

public void setId**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public String getUserName**()** **{**

**return** userName**;**

**}**

public void setUserName**(**String userName**)** **{**

**this.**userName **=** userName**;**

**}**

public Address getAddress**()** **{**

**return** address**;**

**}**

public void setAddress**(**Address address**)** **{**

**this.**address **=** address**;**

**}**

**}**

---------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**one\_to\_one**;**

**import** java**.**util**.**List**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** ge**.**ufc**.**jpa**.**HibernateUtil**;**

public class Test **{**

private static SessionFactory sessionFactory**;**

private Session session**;**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

sessionFactory **=** HibernateUtil**.**getSessionFactory**();**

**}**

**{**

session **=** sessionFactory**.**openSession**();**

session**.**beginTransaction**();**

**}**

public static void main**(**String**[]** args**)** **{**

Test test **=** **null;**

**try** **{**

test **=** **new** Test**();**

test**.**givenData\_whenInsert\_thenCreates1to1relationship**();**

**}** **finally** **{**

test**.**session**.**close**();**

sessionFactory**.**close**();**

**}**

**}**

private void givenData\_whenInsert\_thenCreates1to1relationship**()** **{**

User user **=** **new** User**();**

user**.**setUserName**(**"alice@baeldung.com"**);**

Address address **=** **new** Address**();**

address**.**setStreet**(**"FK Street"**);**

address**.**setCity**(**"FK City"**);**

address**.**setUser**(**user**);**

user**.**setAddress**(**address**);**

/\*

\* Address entry will automatically be created by hibernate, since cascade type

\* is specified as ALL

\*/

session**.**persist**(**user**);**

session**.**getTransaction**().**commit**();**

assert1to1InsertedData**();**

**}**

private void assert1to1InsertedData**()** **{**

@SuppressWarnings**(**"unchecked"**)**

List**<**User**>** userList **=** session**.**createQuery**(**"FROM User"**).**list**();**

System**.**out**.**println**(**userList**.**size**());** // 1

User user **=** userList**.**get**(**0**);**

System**.**out**.**println**(**user**.**getUserName**());** // alice@baeldung.com

Address address **=** user**.**getAddress**();**

System**.**out**.**println**(**address**.**getStreet**());** // FK Street

System**.**out**.**println**(**address**.**getCity**());** // FK City

**}**

**}**

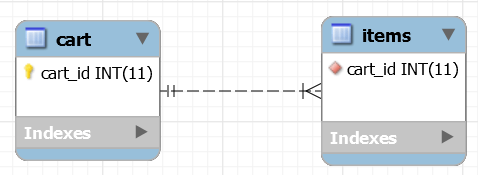
## Hibernate One to Many Annotation

ამ ნაწილში ვისწავლით კავშირს ერთი ბევრთან JPA-ს ანოტაციების გამოყენებით.

### Description

მარტივად რომ ვთქვათ, კავშირი ერთი ბევრთან ნიშნავს, რომ ერთ ჩანაწერს ცხრილში შეესაბამება ბევრი ჩანაწერი სხვა ცხრილში.

დავაკვირდეთ EM დიაგრამას:



cart ცხრილში შენახული გავქვს კალათები, ხოლო items ცხრილში კალათების შესაბამისი ელემენტები. ერთ კალათაში შესაძლოა იყოს ბევრი ელემენტი, ამიტომ გვაქვს კავშირი ერთი ბევრთან. ბაზის დონეზე რომ ვთქვათ, cart\_id არის პირველადი გასაღები cart ცხრილში, ხოლო items ცხრილში გარე გასაღები.

Cart კლასში აღვწეროთ Item-ების სიმრავლე:

public class Cart **{**

//...

@OneToMany**(**mappedBy**=**"cart"**)**

private Set**<**Item**>** items**;**

//...

**}**

ჩვენ ასევე შეგვიძლია დავამატოთ შესაბამის Cart-ზე მითითება Item კლასში @ManyToOne ანოტაციის დაწერით, რაც კავშირს გახდის ორმხრივს - შეგვეძლება წვდომა Cart-დან Item-ზე და ასევე Item-დან Cart-ზე.

mappedBy ატრიბუტი აღნიშნავს, რომ Cart **არ** არის კავშირის მფლობელი მხარე.

### The Models

მოდელ კლასები JPA-ს ანოტაციებით ასე გამოიყურებიან:

@Entity

@Table**(**name**=**"CART"**)**

public class Cart **{**

//...

@OneToMany**(**mappedBy**=**"cart"**)**

private Set**<**Item**>** items**;**

// getters and setters

**}**

mappedBy ატრიბუტში არსებული მნიშვნელობა cart უნდა ერქვას კავშირის ველს Item კლასში:

@Entity

@Table**(**name**=**"ITEMS"**)**

public class Item **{**

//...

@ManyToOne

@JoinColumn**(**name**=**"cart\_id"**,** nullable**=false)**

private Cart cart**;**

public Item**()** **{}**

// getters and setters

**}**

@ManyToOne ანოტაცია იწერება Cart ტიპის ველზე. როგორც წესი, კავშირის მფლობელი უნდა იყოს ის კლასი, რომელშიც გამოყენებულია @ManyToOne ანოტაცია ანუ ბევრის მხარე, ამ შემთხვევაში Cart. @JoinColumn აღწერს item ცხრილში ბმის ველის (FK - გარე გასაღები) სახელს.

### In Action

ვნახოთ გამოძახების მაგალითი:

sessionFactory **=** HibernateAnnotationUtil**.**getSessionFactory**();**

session **=** sessionFactory**.**getCurrentSession**();**

System**.**out**.**println**(**"Session created"**);**

tx **=** session**.**beginTransaction**();**

session**.**save**(**cart**);**

session**.**save**(**item1**);**

session**.**save**(**item2**);**

tx**.**commit**();**

System**.**out**.**println**(**"Cart ID=" **+** cart**.**getId**());**

System**.**out**.**println**(**"item1 ID=" **+** item1**.**getId**()**

**+** ", Foreign Key Cart ID=" **+** item**.**getCart**().**getId**());**

System**.**out**.**println**(**"item2 ID=" **+** item2**.**getId**()**

**+** ", Foreign Key Cart ID=" **+** item**.**getCart**().**getId**());**

გვექნება მსგავსი შედეგი:

Session created

Hibernate**:** insert into CART values **()**

Hibernate**:** insert into ITEMS **(**cart\_id**)**

values **(?)**

Hibernate**:** insert into ITEMS **(**cart\_id**)**

values **(?)**

Cart ID**=**7

item1 ID**=**11**,** Foreign Key Cart ID**=**7

item2 ID**=**12**,** Foreign Key Cart ID**=**7

Closing SessionFactory

### The @ManyToOne Annotation

ორმხრივმა კავშირმა შესაძლოა გამოიწვიოს შეუსაბამობის შეცდომები. მაგალითად, თუ გვსურს, item1 დავამატოთ cart1-ში, ხოლო item2 - cart2-ში, მაგრამ შეცდომით cart1-ის სიაშიც დავამატეთ და შედეგად წარმოიქმნა შეუსაბამობა:

Cart cart1 **=** **new** Cart**();**

Cart cart2 **=** **new** Cart**();**

Item item1 **=** **new** Item**(**cart1**);**

Item item2 **=** **new** Item**(**cart2**);**

Set**<**Item**>** itemsSet **=** **new** HashSet**<**Item**>();**

itemsSet**.**add**(**item1**);**

itemsSet**.**add**(**item2**);**

cart1**.**setItems**(**itemsSet**);** // wrong!

აქ item2 მიუთითებს cart2-ზე, მაგრამ cart2-ს არ აქვს მითითება item2-ზე. როგორ უნდა შეინახოს item2 ბაზაში Hibernate-მა? item2-ის გარე გასაღები cart1-ის იქნება თუ cart2-ის?

ჩვენ შეგვიძლია გადავწყვიტოთ ეს ორაზროვნება კავშირის მფლობელის შემოღებით. უპირატესობა მიენიჭება მფლობელი მხარის მითითებებს და ისინი შეინახება ბაზაშიც.

### Item as the Owning Side

როგორც უკვე აღვნიშნეთ, JPA სპეციფიკაციისf მიხედვით, კარგი პრაქტიკაა many-to-one მხარე აღვწეროთ მიმართების მფლობელ მხარედ.

შესაბამისად, Item იქნება მფლობელი მხარე, ხოლო Cart საპირისპირო მხარე. mappedBy ატრიბუტის გამოყენებით Cart კლასში, მას აღვწერთ, როგორც საპირისპირო მხარეს. @ManyToOne ანოტაცია კი გამოყენებული გავქვს Item კლასში, რაც მას ხდის მიმართების მფლობელ მხარედ.

რომ დავუბრუნდეთ ზემოთ მოყვანილ შეუსაბამობის მაგალითს, Hibernate-მა იცის რომ item2-ზე მითითება არის უფრო მნიშვნელოვანი და ბაზაშიც item2-ს შეინახავს.

ვნახოთ შედეგი:

item1 ID**=**1**,** Foreign Key Cart ID**=**1

item2 ID**=**2**,** Foreign Key Cart ID**=**2

მიუხედავად იმისა, რომ cart1-ის item-ების სიმრავლეში დავამატეთ item2-იც, ბაზის ცხრილში item2 მიება cart2-ზე, რადგან item არის მიმართების მფლობელი მხარე.

### Cart as the Owning Side

შესაძლოა ასევე პირიქით, one-to-many გავხადოთ მფლობელი მხარე, ხოლო many-to-one აღმოჩნდება საპირისპირო მხარეს.

მიუხედავად იმისა, რომ ეს არ არის კარგი პრაქტიკა, ვცადოთ მაინც.

public class ItemOIO **{**

// ...

@ManyToOne

@JoinColumn**(**name **=** "cart\_id"**,** insertable **=** **false,** updatable **=** **false)**

private CartOIO cart**;**

//..

**}**

public class CartOIO **{**

//..

@OneToMany

@JoinColumn**(**name **=** "cart\_id"**)** // we need to duplicate the physical information

private Set**<**ItemOIO**>** items**;**

//..

**}**

შეამჩნევდით, რომ ამოვიღეთ mappedBy ატრიბუტი და ასევე many-to-one მხარის @JoinColumn გავხადეთ insertable = false და updatable = false.

ზემოთ მოყვანილი მაგალითი კი უკვე ასეთ შედეგს მოგვცემს:

item1 ID**=**1**,** Foreign Key Cart ID**=**1

item2 ID**=**2**,** Foreign Key Cart ID**=**1

როგორც ხედავთ, item2 უკვე cart1-ში ჩავარდა.

გასატესტად ბაზაში დავამატოთ ცხრილები:

**CREATE** **TABLE** CART **(**cart\_id INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY** **(START** **WITH** 10**,** **INCREMENT** **BY** 1**));**

**CREATE** **TABLE** ITEMS **(**id INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY** **(START** **WITH** 100**,** **INCREMENT** **BY** 1**),** cart\_id INTEGER **NOT** **NULL,** **FOREIGN** **KEY(**cart\_id**)** **REFERENCES** CART**(**cart\_id**));**

**CREATE** **TABLE** CARTOIO **(**cart\_id INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY** **(START** **WITH** 20**,** **INCREMENT** **BY** 1**));**

**CREATE** **TABLE** ITEMSOIO **(**id INTEGER **NOT** **NULL** **PRIMARY** **KEY** GENERATED ALWAYS **AS** **IDENTITY** **(START** **WITH** 200**,** **INCREMENT** **BY** 1**),** cart\_id INTEGER**,** **FOREIGN** **KEY(**cart\_id**)** **REFERENCES** CARTOIO**(**cart\_id**));**

hibernate.cfg.xml ფაილში დავამატოთ:

<mapping class=**"ge.ufc.jpa.one\_to\_many.Cart"**/>

<mapping class=**"ge.ufc.jpa.one\_to\_many.Item"**/>

<mapping class=**"ge.ufc.jpa.one\_to\_many.CartOIO"**/>

<mapping class=**"ge.ufc.jpa.one\_to\_many.ItemOIO"**/>

პროექტში კი დავამატოთ კლასები:

package ge**.**ufc**.**jpa**.**one\_to\_many**;**

**import** java**.**util**.**Set**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**GeneratedValue**;**

**import** javax**.**persistence**.**GenerationType**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**OneToMany**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "CART"**)**

public class Cart **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

@Column**(**name **=** "cart\_id"**)**

private long id**;**

@OneToMany**(**mappedBy **=** "cart"**)**

private Set**<**Item**>** items**;**

public long getId**()** **{**

**return** id**;**

**}**

public void setId**(**long id**)** **{**

**this.**id **=** id**;**

**}**

public Set**<**Item**>** getItems**()** **{**

**return** items**;**

**}**

public void setItems**(**Set**<**Item**>** items**)** **{**

**this.**items **=** items**;**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**one\_to\_many**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**GeneratedValue**;**

**import** javax**.**persistence**.**GenerationType**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**JoinColumn**;**

**import** javax**.**persistence**.**ManyToOne**;**

**import** javax**.**persistence**.**Table**;**

@Entity**(**name **=** "ITEM"**)**

@Table**(**name **=** "ITEMS"**)**

public class Item **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

@Column**(**name **=** "id"**)**

private long id**;**

@ManyToOne

@JoinColumn**(**name **=** "cart\_id"**,** nullable **=** **false)**

private Cart cart**;**

// Hibernate requires no-args constructor

public Item**()** **{**

**}**

public Item**(**Cart c**)** **{**

**this.**cart **=** c**;**

**}**

public Cart getCart**()** **{**

**return** cart**;**

**}**

public void setCart**(**Cart cart**)** **{**

**this.**cart **=** cart**;**

**}**

public long getId**()** **{**

**return** id**;**

**}**

public void setId**(**long id**)** **{**

**this.**id **=** id**;**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**one\_to\_many**;**

**import** java**.**util**.**Set**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**GeneratedValue**;**

**import** javax**.**persistence**.**GenerationType**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**JoinColumn**;**

**import** javax**.**persistence**.**OneToMany**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "CARTOIO"**)**

public class CartOIO **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

@Column**(**name **=** "cart\_id"**)**

private long id**;**

@OneToMany

@JoinColumn**(**name **=** "cart\_id"**)** // we need to duplicate the physical information

private Set**<**ItemOIO**>** items**;**

public long getId**()** **{**

**return** id**;**

**}**

public void setId**(**long id**)** **{**

**this.**id **=** id**;**

**}**

public Set**<**ItemOIO**>** getItems**()** **{**

**return** items**;**

**}**

public void setItems**(**Set**<**ItemOIO**>** items**)** **{**

**this.**items **=** items**;**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**one\_to\_many**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**GeneratedValue**;**

**import** javax**.**persistence**.**GenerationType**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**JoinColumn**;**

**import** javax**.**persistence**.**ManyToOne**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "ITEMSOIO"**)**

public class ItemOIO **{**

@Id

@GeneratedValue**(**strategy **=** GenerationType**.**IDENTITY**)**

@Column**(**name **=** "id"**)**

private long id**;**

@ManyToOne

@JoinColumn**(**name **=** "cart\_id"**,** insertable **=** **false,** updatable **=** **false)**

private CartOIO cart**;**

// Hibernate requires no-args constructor

public ItemOIO**()** **{**

**}**

public ItemOIO**(**CartOIO c**)** **{**

**this.**cart **=** c**;**

**}**

public CartOIO getCartOIO**()** **{**

**return** cart**;**

**}**

public void setCartOIO**(**CartOIO cart**)** **{**

**this.**cart **=** cart**;**

**}**

public long getId**()** **{**

**return** id**;**

**}**

public void setId**(**long id**)** **{**

**this.**id **=** id**;**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**one\_to\_many**;**

**import** java**.**util**.**HashSet**;**

**import** java**.**util**.**Set**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** org**.**hibernate**.**Transaction**;**

**import** ge**.**ufc**.**jpa**.**HibernateUtil**;**

public class HibernateOneToManyAnnotationMain **{**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

**}**

public static void main**(**String**[]** args**)** **{**

Cart cart **=** **new** Cart**();**

Item item1 **=** **new** Item**(**cart**);**

Item item2 **=** **new** Item**(**cart**);**

Set**<**Item**>** itemsSet **=** **new** HashSet**<>();**

itemsSet**.**add**(**item1**);**

itemsSet**.**add**(**item2**);**

cart**.**setItems**(**itemsSet**);**

SessionFactory sessionFactory **=** HibernateUtil**.**getSessionFactory**();**

Session session **=** sessionFactory**.**openSession**();**

System**.**out**.**println**(**"Session created"**);**

**try** **{**

// start transaction

Transaction tx **=** session**.**beginTransaction**();**

// Save the Model object

session**.**save**(**cart**);**

session**.**save**(**item1**);**

session**.**save**(**item2**);**

// Commit transaction

tx**.**commit**();**

System**.**out**.**println**(**"Cart ID=" **+** cart**.**getId**());**

System**.**out**.**println**(**"item1 ID=" **+** item1**.**getId**()** **+** ", Foreign Key Cart ID=" **+** item1**.**getCart**().**getId**());**

System**.**out**.**println**(**"item2 ID=" **+** item2**.**getId**()** **+** ", Foreign Key Cart ID=" **+** item2**.**getCart**().**getId**());**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}** **finally** **{**

**if** **(!**sessionFactory**.**isClosed**())** **{**

System**.**out**.**println**(**"Closing SessionFactory"**);**

sessionFactory**.**close**();**

**}**

**}**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**one\_to\_many**;**

**import** java**.**util**.**HashSet**;**

**import** java**.**util**.**Set**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** org**.**hibernate**.**Transaction**;**

**import** ge**.**ufc**.**jpa**.**HibernateUtil**;**

public class HibernateManyIsOwningSide **{**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

**}**

public static void main**(**String**[]** args**)** **{**

Cart cart1 **=** **new** Cart**();**

Cart cart2 **=** **new** Cart**();**

Item item1 **=** **new** Item**(**cart1**);**

Item item2 **=** **new** Item**(**cart2**);**

Set**<**Item**>** itemsSet **=** **new** HashSet**<>();**

itemsSet**.**add**(**item1**);**

itemsSet**.**add**(**item2**);**

cart1**.**setItems**(**itemsSet**);**

SessionFactory sessionFactory **=** HibernateUtil**.**getSessionFactory**();**

Session session **=** sessionFactory**.**openSession**();**

System**.**out**.**println**(**"Session created"**);**

**try** **{**

// start transaction

Transaction tx **=** session**.**beginTransaction**();**

// Save the Model object

session**.**save**(**cart1**);**

session**.**save**(**cart2**);**

session**.**save**(**item1**);**

session**.**save**(**item2**);**

// Commit transaction

tx**.**commit**();**

session**.**close**();**

session **=** sessionFactory**.**openSession**();**

item1 **=** session**.**get**(**Item**.**class**,** item1**.**getId**());**

item2 **=** session**.**get**(**Item**.**class**,** item2**.**getId**());**

System**.**out**.**printf**(**"item1 ID=%d, Foreign Key Cart ID=%d%n"**,** item1**.**getId**(),** item1**.**getCart**().**getId**());**

System**.**out**.**printf**(**"item2 ID=%d, Foreign Key Cart ID=%d%n"**,** item2**.**getId**(),** item2**.**getCart**().**getId**());**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}** **finally** **{**

**if** **(!**sessionFactory**.**isClosed**())** **{**

System**.**out**.**println**(**"Closing SessionFactory"**);**

sessionFactory**.**close**();**

**}**

**}**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**one\_to\_many**;**

**import** java**.**util**.**HashSet**;**

**import** java**.**util**.**Set**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** org**.**hibernate**.**Transaction**;**

**import** ge**.**ufc**.**jpa**.**HibernateUtil**;**

public class HibernateOneIsOwningSide **{**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

**}**

public static void main**(**String**[]** args**)** **{**

CartOIO cart1 **=** **new** CartOIO**();**

CartOIO cart2 **=** **new** CartOIO**();**

ItemOIO item1 **=** **new** ItemOIO**(**cart1**);**

ItemOIO item2 **=** **new** ItemOIO**(**cart2**);**

Set**<**ItemOIO**>** itemsSet **=** **new** HashSet**<>();**

itemsSet**.**add**(**item1**);**

itemsSet**.**add**(**item2**);**

cart1**.**setItems**(**itemsSet**);**

SessionFactory sessionFactory **=** HibernateUtil**.**getSessionFactory**();**

Session session **=** sessionFactory**.**openSession**();**

System**.**out**.**println**(**"Session created"**);**

**try** **{**

// start transaction

Transaction tx **=** session**.**beginTransaction**();**

// Save the Model object

session**.**save**(**cart1**);**

session**.**save**(**cart2**);**

session**.**save**(**item1**);**

session**.**save**(**item2**);**

// Commit transaction

tx**.**commit**();**

session**.**close**();**

session **=** sessionFactory**.**openSession**();**

item1 **=** session**.**get**(**ItemOIO**.**class**,** item1**.**getId**());**

item2 **=** session**.**get**(**ItemOIO**.**class**,** item2**.**getId**());**

System**.**out**.**printf**(**"item1 ID=%d, Foreign Key CartOIO ID=%d%n"**,** item1**.**getId**(),** item1**.**getCartOIO**().**getId**());**

System**.**out**.**printf**(**"item2 ID=%d, Foreign Key CartOIO ID=%d%n"**,** item2**.**getId**(),** item2**.**getCartOIO**().**getId**());**

**}** **catch** **(**Exception e**)** **{**

e**.**printStackTrace**();**

**}** **finally** **{**

**if** **(!**sessionFactory**.**isClosed**())** **{**

System**.**out**.**println**(**"Closing SessionFactory"**);**

sessionFactory**.**close**();**

**}**

**}**

**}**

**}**

## Many-To-Many Relationship in JPA

ამ ნაწილში JPA-ში განვიხილავთ კავშირს ბევრი ბევრთან. გამოვიყენებთ სტუდენტების და კურსების მოდელს და მათ შორის კავშირის დამყარების სხვადასხვა გზებს.

### Basic Many-to-Many

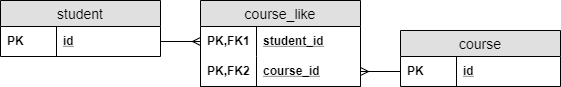
#### Modeling a Many-to-Many Relationship

ბევრი ბევრთან კავშირში თითოეული მხარე მეორე მხარესთან დაკავშირებულია ერთზე მეტი ეგზემპლარით.

განვიხილოთ სტუდენტებს მაგალითი, რომლებიც ირჩევენ კურსებს. ერთ სტუდენტს შეუძლია აირჩიოს ბევრი კურსი და ერთ კურსი შეიძლება აირჩიოს ბევრმა სტუდენტმა.

https://www.baeldung.com/wp-content/uploads/2018/11/simple-er.png

RDBMS-ში ჩვენ შეგვიძლია კავშირი დავამყაროთ გარე გასაღებებით. ბევრი ბევრთან კავშირის დროს დამატებით გვჭირდება შუალედური ცხრილის შექმნა, რომელსაც ორივე ცხრილი დაუკავშირდება გარე გასაღებით.



როგორც წესი, დამაკავშირებელ შუალედურ ცხრილში გარე გასაღებების კომბინაცია წარმოადგენს თავად ამ ცხრილის პირველად გასაღებს.

#### Implementation in JPA

ბევრი ბევრთან კავშირის მოდელირება POJO კლასებში მარტივია: თითოეულ კლასში ველად უნდა აღვწეროთ კოლექცია, რომლის ელემენტების ტიპი იქნება მეორე კლასი.

ორივე კლასს თავისთავად უნდა ჰქონდეთ @Entity და @Id ანოტაციები.

კოლექციები კი უნდა მოვნიშნოთ @ManyToMany ანოტაციებით:

@Entity

class Student **{**

@Id

Long id**;**

@ManyToMany

Set**<**Course**>** likedCourses**;**

// additional properties

// standard constructors, getters, and setters

**}**

@Entity

class Course **{**

@Id

Long id**;**

@ManyToMany

Set**<**Student**>** likes**;**

// additional properties

// standard constructors, getters, and setters

**}**

ახლა კი უნდა ავირჩიოთ მიმართების მფლობელი მხარე. ამ შემთხვევაში ავირჩიოთ Student. ამისთვის უნდა გამოვიყენოთ @JoinTable ანოტაცია Student კლასში. უნდა მივუთითოთ დამაკავშირებელი ცხრილის სახელი (course\_like) და მასში არსებული გარე გასაღებები @JoinColumn ანოტაციით გავწეროთ. joinColumns ატრიბუტში გაწერილია შუალედური ცხრილის კავშირის მფლობელ მხარესთან დამაკავშირებელი ველ(ებ)ი (გარე გასაღები), ხოლო inverseJoinColumns -ში საპირისპირო მხარეს მდგომ ცხრილთან დაკავშირებული გარე გასაღებ(ებ)ი.

@ManyToMany

@JoinTable**(**

name **=** "course\_like"**,**

joinColumns **=** @JoinColumn**(**name **=** "student\_id"**),**

inverseJoinColumns **=** @JoinColumn**(**name **=** "course\_id"**))**

Set**<**Course**>** likedCourses**;**

მეორე მხარეს კი (კავშირის **არა-მფლობელ** მხარეს) mappedBy ატრიბუტში უნდა მივუთითო კავშირის ველის სახელი მფლობელ კლასში:

@ManyToMany**(**mappedBy **=** "likedCourses"**)**

Set**<**Student**>** likes**;**

### Many-to-Many Using a Composite Key

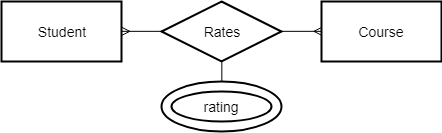
### Modeling Relationship Attributes

ვთქვათ გვსურს, გავაკეთოთ სტუდენტების შეფასება კურსების მიხედვით. ერთ სტუდენტს შეუძლია აირჩიოს ბევრი კურსი და ერთ კურსი შეიძლება აირჩიოს ბევრმა სტუდენტმა. კავშირი გვაქვს ბევრი ბევრთან.

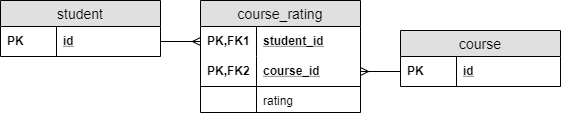
ამ კავშირს ართულებს ის მომენტი, რომ შეიძლება გვქონდეს კავშირის ველების გარდა სხვა ველებიც, რომლებიც დააკავშირებს სტუდენტებსა და კურსებს. მაგალითად, ქულა (შეფასება), რომელმა სტუდეტნმა რომელ კურსზე რა ქულა მიიღო.

სად უნდა შევინახოთ ეს ინფორმაცია? ჩვენ არ შეგვიძლია ის შევინახოთ Student-ის entity-ში, რადგან სხვადასხვა საგანში შეიძლება სტუდენტს სხვადასხვა ქულა ჰქონდეს. ასევე ანალოგიურად არ შეგვიძლია ამ ინფორმაციის შენახვა Course-ის entity-ში.

ეს არის სიტუაცია, როდესაც რელაციურ კავშირს აქვს დამატებითი ატრიბუტ(ებ)ი. დავაკვირდეთ ER დიაგრამას:



მისი მოდელირება ცხრილებში შეგვიძლია ბევრი ბევრთან კავშირის თითქმის იმავე გზით - შუალედური გადამკვეთი ცხრილით. სხვაობა არის მხოლოდ ის, რომ შუალედურ ცხრილს დაემატება ახალი ველი rating:



#### Creating a Composite Key in JPA

წინა მაგალითში ბევრი ბევრთან კავშირის განხორციელება მარტივი იყო - Entity-ები პირდაპირ დავაკავშირეთ ერთმანეთთან. ახლა გვაქვს პრობლემა, სად შევინახოთ კავშირის მახასიათებელი ველი.

კავშირისთვის უკვე დაგვჭირდება დამატებითი Entity კლასის შექმნა:

რა თქმა უნდა, ყველა JPA entity-ს სჭირდება პირველადი გასაღები (Primary Key). რადგან ჩვენს შემთხვევაში პირველადი გასაღები კომპოზიტურია (შედგება რამდენიმე ველისგან), პირველ რიგში მოგვიწევს შევქმნათ ახალი კლასი, რომელშიც აღვწერთ ამ ველებს:

@Embeddable

class CourseRatingKey **implements** Serializable **{**

@Column**(**name **=** "student\_id"**)**

Long studentId**;**

@Column**(**name **=** "course\_id"**)**

Long courseId**;**

// standard constructors, getters, and setters

// hashcode and equals implementation

**}**

მიაქციეთ ყურადღება, რომ კლასი უნდა იყოს Serializable და მონიშნული @Embeddable ანოტაციით.

#### Using a Composite Key in JPA

ახლა კი აღვწეროთ დამაკავშირებელი ცხრილის Entity კლასი:

@Entity

class CourseRating **{**

@EmbeddedId

CourseRatingKey id**;**

@ManyToOne

@MapsId**(**"studentId"**)**

@JoinColumn**(**name **=** "student\_id"**)**

Student student**;**

@ManyToOne

@MapsId**(**"courseId"**)**

@JoinColumn**(**name **=** "course\_id"**)**

Course course**;**

int rating**;**

// standard constructors, getters, and setters

**}**

ეს კლასი მარტივად გამოიყურება. დავაკვირდეთ მნიშვნელოვან მომენტებს:

* პირველადი გასაღების მოსანიშნად გამოვიყენეთ @EmbeddedId ანოტაცია. ველის ტიპი კი არის CourseRatingKey.
* student და course ველები აღვწერეთ @MapsId ანოტაციით.

@MapsId ნიშნავს, რომ ჩვენ ვუკავშირებთ ამ ველებს გასაღების ნაწილს და ისინი წარმოადგენენ ბევრი ერთთან კავშირის გასაღებებს. ჩვენ ეს გვჭირდება, რადგან კომპოზიტური პირველადი გასაღების CourseRatingKey კლასში შეუძლებელია Entity-ების აღწერა და გვიწევს მათი, ასე ვთქვათ ხელმეორედ აღწერა CourseRating კლასში და CourseRatingKey კლასს ვუკავშირებთ @MapsId ანოტაციით, რომლის value ატრიბუტში მითითებული გვაქვს სწორედ CourseRatingKey კლასის შესაბამისი ველების სახელები.

Student და Course entity-ები კი უკვე ასე მარტივად გამოიყურებიან:

class Student **{**

// ...

@OneToMany**(**mappedBy **=** "student"**)**

Set**<**CourseRating**>** ratings**;**

// ...

**}**

class Course **{**

// ...

@OneToMany**(**mappedBy **=** "course"**)**

Set**<**CourseRating**>** ratings**;**

// ...

**}**

როგოც ხედავთ, რეალურად ბევრი ბევრთან კავშირი დავიყვანეთ ერთი ერთთან კავშირებზე ორი მიმართულებით.

### Many-to-Many With a New Entity

#### Modeling Relationship Attributes

გადამკვეთ ცხრილში შეიძლება კიდევ დაგვჭირდეს სხვა ველებიც: მაგალითად, როდის დარეგისტრირდა სტუდენტი კურსზე და რა ხარისხი (შეფასება, grade) მიიღო.

შეგვიძლია გამოვიყენოთ ისევ კომპოზიტური პირველადი გასაღებით გადაწყვეტა, მაგრამ აქ ჩნდება კიდევ ერთი პრობლემა. სტუდენტებმა შესაძლოა ერთი კურსი ვერ გაიარონ ერთი ცდით და რამდენჯერმე გავლა დასჭირდეთ.

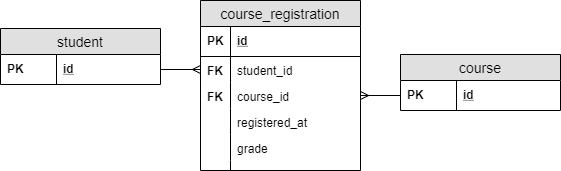
ამ შემთხვევაში გვიჩნდება ერთი და იგივე სტუდენტ-კურსის ერთზე მეტი კავშირი. შესაბამისად, გადამკვეთ ცხრილში გვექნება ერთზე მეტი ჩანაწერი ერთნაირი student\_id-course\_id წყვილებით. ჩვენ კი წინა გადაწყვეტებში ამ წყვილზე გვედო უნიკალური პირველადი გასაღები. შესაბამისად, აქ უკვე სხვა ცალკე პირველადი გასაღები დაგვჭირდება, რადგან წარმოიქმნა დუბლირება გარე გასაღებების წყვილში.

შემოვიღოთ ახალი Entity, რომელიც შეინახავს რეგისტრაციის დეტალებს:

https://www.baeldung.com/wp-content/uploads/2018/11/relation-entity-er-updated.png

Registration entity წარმოადგენს ორ Entity-ს შორის კავშირს და მას ექნება უკვე თავისი ცალკე პირველადი გასაღები. წინა მაგალითებში ჩვენ ვიყენებდით კომპოზიტურ პირველად გასაღებს, რომელიც ორი გარე გასაღებისგან შედგებოდა, თუმცა აქ აღარ გამოგვადგება.

ამჯერად გარე გასღებები ისევ დარჩებიან გარე გასაღებებად, მაგრამ ვეღარ იქნებიან პირველადი გასაღების ნაწილები:



#### Implementation in JPA

შევქმნათ დამაკავშირებელი course\_registration ცხრილის შესაბამისი Entity:

@Entity

class CourseRegistration **{**

@Id

Long id**;**

@ManyToOne

@JoinColumn**(**name **=** "student\_id"**)**

Student student**;**

@ManyToOne

@JoinColumn**(**name **=** "course\_id"**)**

Course course**;**

LocalDateTime registeredAt**;**

int grade**;**

// additional properties

// standard constructors, getters, and setters

**}**

კავშირების განსაზღვრა კვლავაც გვჭირდება Student და Course კლასებში:

class Student **{**

// ...

@OneToMany**(**mappedBy **=** "student"**)**

Set**<**CourseRegistration**>** registrations**;**

// ...

**}**

class Course **{**

// ...

@OneToMany**(**mappedBy **=** "course"**)**

Set**<**CourseRegistration**>** registrations**;**

// ...

**}**

რა თქმა უნდა, ჩვენ შეგვეძლო გამოგვეყენებინა ეს გადაწყვეტა წინა მაგალითშიც, მაგრამ გამოყოფილი ცალკე პირველადი გასაღების შექმნა, თუ იძულებული არ ვართ, მაინც არასაჭიროდ გამოიყურება. გარდა ამისა, ამ კომპოზიციურ გასაღებს ჰქონდა მკაფიო მნიშვნელობა: რომელ ერთეულებს ვუკავშირდებოდით მიმართებაში.

თუმცა საბოლოო არჩევანი ამ ორ გადაწყვეტას შორის არის ინდივიდუალური.

## Conclusion

ამ ლექციაში ვისწვალეთ, როგორ ავაწყოთ JPA-ში ბევრი ბევრთან კავშირი RDBMS-ის გამოყენებით. განვიხილეთ გადაწყვეტის სამი გზა. თითოეულს ჰქონდა როგორც უპირატესობა, ისე ნაკლოვანებები, რომლებიც განისაზღვრებიან სხვადასხვა ასპექტებით:

* კოდის სიცხადე.
* ბაზის სიცხადე.
* კავშირისთვის ატრიბუტების მინიჭება.
* რამდენი Entity შეგვიძლია დავაკავშიროთ.
* შეგვეძლოს ერთნაირი Entity-ების რამდენჯერმე დაკავშირება.

გასატესტად ბაზაში დავამატოთ ცხრილები:

**CREATE** **TABLE** course **(**id bigint **NOT** **NULL,** **PRIMARY** **KEY** **(**id**));**

**CREATE** **TABLE** student **(**id bigint **NOT** **NULL,** **PRIMARY** **KEY** **(**id**));**

**CREATE** **TABLE** course\_like **(**student\_id bigint **NOT** **NULL,** course\_id bigint **NOT** **NULL,**

**PRIMARY** **KEY** **(**student\_id**,** course\_id**),**

**FOREIGN** **KEY** **(**student\_id**)** **REFERENCES** student **(**id**),**

**FOREIGN** **KEY** **(**course\_id**)** **REFERENCES** course **(**id**)**

**);**

**CREATE** **TABLE** course\_rating **(**course\_id bigint **NOT** **NULL,** student\_id bigint **NOT** **NULL,** rating int **NOT** **NULL,**

**PRIMARY** **KEY** **(**course\_id**,** student\_id**),**

**FOREIGN** **KEY** **(**student\_id**)** **REFERENCES** student **(**id**),**

**FOREIGN** **KEY** **(**course\_id**)** **REFERENCES** course **(**id**)**

**);**

**CREATE** **TABLE** course\_registration **(**id bigint **NOT** **NULL,** grade int**,** registered\_at TIMESTAMP **NOT** **NULL,**

course\_id bigint **NOT** **NULL,** student\_id bigint **NOT** **NULL,**

**PRIMARY** **KEY** **(**id**),**

**FOREIGN** **KEY** **(**student\_id**)** **REFERENCES** student **(**id**),**

**FOREIGN** **KEY** **(**course\_id**)** **REFERENCES** course **(**id**)**

**);**

hibernate.cfg.xml ფაილში ჩავამატოთ:

<mapping class=**"ge.ufc.jpa.many\_to\_many.Student"**/>

<mapping class=**"ge.ufc.jpa.many\_to\_many.Course"**/>

<mapping class=**"ge.ufc.jpa.many\_to\_many.CourseRatingKey"**/>

<mapping class=**"ge.ufc.jpa.many\_to\_many.CourseRating"**/>

<mapping class=**"ge.ufc.jpa.many\_to\_many.CourseRegistration"**/>

პროექტს კი დავამატოთ კლასები:

package ge**.**ufc**.**jpa**.**many\_to\_many**;**

**import** java**.**util**.**HashSet**;**

**import** java**.**util**.**Set**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**JoinTable**;**

**import** javax**.**persistence**.**JoinColumn**;**

**import** javax**.**persistence**.**ManyToMany**;**

**import** javax**.**persistence**.**OneToMany**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "student"**)**

public class Student **{**

@Id

@Column**(**name **=** "id"**)**

private Long id**;**

@ManyToMany

@JoinTable**(**name **=** "course\_like"**,** joinColumns **=** @JoinColumn**(**name **=** "student\_id"**),** inverseJoinColumns **=** @JoinColumn**(**name **=** "course\_id"**))**

private Set**<**Course**>** likedCourses **=** **new** HashSet**<>();**

@OneToMany**(**mappedBy **=** "student"**)**

private Set**<**CourseRating**>** ratings **=** **new** HashSet**<>();**

@OneToMany**(**mappedBy **=** "student"**)**

private Set**<**CourseRegistration**>** registrations **=** **new** HashSet**<>();**

// additional properties

public Student**()** **{**

**}**

public Student**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public Long getId**()** **{**

**return** id**;**

**}**

public void setId**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public Set**<**Course**>** getLikedCourses**()** **{**

**return** likedCourses**;**

**}**

public void setLikedCourses**(**Set**<**Course**>** likedCourses**)** **{**

**this.**likedCourses **=** likedCourses**;**

**}**

public Set**<**CourseRating**>** getRatings**()** **{**

**return** ratings**;**

**}**

public void setRatings**(**Set**<**CourseRating**>** ratings**)** **{**

**this.**ratings **=** ratings**;**

**}**

public Set**<**CourseRegistration**>** getRegistrations**()** **{**

**return** registrations**;**

**}**

public void setRegistrations**(**Set**<**CourseRegistration**>** registrations**)** **{**

**this.**registrations **=** registrations**;**

**}**

@Override

public int hashCode**()** **{**

final int prime **=** 31**;**

int result **=** 1**;**

result **=** prime **\*** result **+** **((**id **==** **null)** **?** 0 **:** id**.**hashCode**());**

**return** result**;**

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(this** **==** obj**)**

**return** **true;**

**if** **(**obj **==** **null)**

**return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())**

**return** **false;**

Student other **=** **(**Student**)** obj**;**

**if** **(**id **==** **null)** **{**

**if** **(**other**.**id **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**id**.**equals**(**other**.**id**))**

**return** **false;**

**return** **true;**

**}**

**}**

----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**many\_to\_many**;**

**import** java**.**util**.**HashSet**;**

**import** java**.**util**.**Set**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**ManyToMany**;**

**import** javax**.**persistence**.**OneToMany**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "course"**)**

public class Course **{**

@Id

@Column**(**name **=** "id"**)**

private Long id**;**

@ManyToMany**(**mappedBy **=** "likedCourses"**)**

private Set**<**Student**>** likes **=** **new** HashSet**<>();**

@OneToMany**(**mappedBy **=** "course"**)**

private Set**<**CourseRating**>** ratings **=** **new** HashSet**<>();**

@OneToMany**(**mappedBy **=** "course"**)**

private Set**<**CourseRegistration**>** registrations **=** **new** HashSet**<>();**

// additional properties

public Course**()** **{**

**}**

public Course**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public Long getId**()** **{**

**return** id**;**

**}**

public void setId**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public Set**<**Student**>** getLikes**()** **{**

**return** likes**;**

**}**

public void setLikes**(**Set**<**Student**>** likes**)** **{**

**this.**likes **=** likes**;**

**}**

public Set**<**CourseRating**>** getRatings**()** **{**

**return** ratings**;**

**}**

public void setRatings**(**Set**<**CourseRating**>** ratings**)** **{**

**this.**ratings **=** ratings**;**

**}**

public Set**<**CourseRegistration**>** getRegistrations**()** **{**

**return** registrations**;**

**}**

public void setRegistrations**(**Set**<**CourseRegistration**>** registrations**)** **{**

**this.**registrations **=** registrations**;**

**}**

@Override

public int hashCode**()** **{**

final int prime **=** 31**;**

int result **=** 1**;**

result **=** prime **\*** result **+** **((**id **==** **null)** **?** 0 **:** id**.**hashCode**());**

**return** result**;**

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(this** **==** obj**)**

**return** **true;**

**if** **(**obj **==** **null)**

**return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())**

**return** **false;**

Course other **=** **(**Course**)** obj**;**

**if** **(**id **==** **null)** **{**

**if** **(**other**.**id **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**id**.**equals**(**other**.**id**))**

**return** **false;**

**return** **true;**

**}**

**}**

----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**many\_to\_many**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Embeddable**;**

**import** java**.**io**.**Serializable**;**

@Embeddable

public class CourseRatingKey **implements** Serializable **{**

private static final long serialVersionUID **=** 1L**;**

@Column**(**name **=** "student\_id"**)**

private Long studentId**;**

@Column**(**name **=** "course\_id"**)**

private Long courseId**;**

public CourseRatingKey**()** **{**

**}**

public CourseRatingKey**(**Long studentId**,** Long courseId**)** **{**

**this.**studentId **=** studentId**;**

**this.**courseId **=** courseId**;**

**}**

public Long getStudentId**()** **{**

**return** studentId**;**

**}**

public void setStudentId**(**Long studentId**)** **{**

**this.**studentId **=** studentId**;**

**}**

public Long getCourseId**()** **{**

**return** courseId**;**

**}**

public void setCourseId**(**Long courseId**)** **{**

**this.**courseId **=** courseId**;**

**}**

@Override

public int hashCode**()** **{**

final int prime **=** 31**;**

int result **=** 1**;**

result **=** prime **\*** result **+** **((**courseId **==** **null)** **?** 0 **:** courseId**.**hashCode**());**

result **=** prime **\*** result **+** **((**studentId **==** **null)** **?** 0 **:** studentId**.**hashCode**());**

**return** result**;**

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(this** **==** obj**)**

**return** **true;**

**if** **(**obj **==** **null)**

**return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())**

**return** **false;**

CourseRatingKey other **=** **(**CourseRatingKey**)** obj**;**

**if** **(**courseId **==** **null)** **{**

**if** **(**other**.**courseId **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**courseId**.**equals**(**other**.**courseId**))**

**return** **false;**

**if** **(**studentId **==** **null)** **{**

**if** **(**other**.**studentId **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**studentId**.**equals**(**other**.**studentId**))**

**return** **false;**

**return** **true;**

**}**

**}**

----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**many\_to\_many**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**EmbeddedId**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**JoinColumn**;**

**import** javax**.**persistence**.**ManyToOne**;**

**import** javax**.**persistence**.**MapsId**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "course\_rating"**)**

public class CourseRating **{**

@EmbeddedId

private CourseRatingKey id**;**

@ManyToOne

@MapsId**(**"studentId"**)**

@JoinColumn**(**name **=** "student\_id"**)**

private Student student**;**

@ManyToOne

@MapsId**(**"courseId"**)**

@JoinColumn**(**name **=** "course\_id"**)**

private Course course**;**

@Column**(**name **=** "rating"**)**

private int rating**;**

public CourseRating**()** **{**

**}**

public CourseRatingKey getId**()** **{**

**return** id**;**

**}**

public void setId**(**CourseRatingKey id**)** **{**

**this.**id **=** id**;**

**}**

public Student getStudent**()** **{**

**return** student**;**

**}**

public void setStudent**(**Student student**)** **{**

**this.**student **=** student**;**

**}**

public Course getCourse**()** **{**

**return** course**;**

**}**

public void setCourse**(**Course course**)** **{**

**this.**course **=** course**;**

**}**

public int getRating**()** **{**

**return** rating**;**

**}**

public void setRating**(**int rating**)** **{**

**this.**rating **=** rating**;**

**}**

@Override

public int hashCode**()** **{**

final int prime **=** 31**;**

int result **=** 1**;**

result **=** prime **\*** result **+** **((**id **==** **null)** **?** 0 **:** id**.**hashCode**());**

**return** result**;**

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(this** **==** obj**)**

**return** **true;**

**if** **(**obj **==** **null)**

**return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())**

**return** **false;**

CourseRating other **=** **(**CourseRating**)** obj**;**

**if** **(**id **==** **null)** **{**

**if** **(**other**.**id **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**id**.**equals**(**other**.**id**))**

**return** **false;**

**return** **true;**

**}**

**}**

----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**many\_to\_many**;**

**import** java**.**time**.**LocalDateTime**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**JoinColumn**;**

**import** javax**.**persistence**.**ManyToOne**;**

**import** javax**.**persistence**.**Table**;**

@Entity

@Table**(**name **=** "course\_registration"**)**

public class CourseRegistration **{**

@Id

@Column**(**name **=** "id"**)**

private Long id**;**

@ManyToOne

@JoinColumn**(**name **=** "student\_id"**)**

private Student student**;**

@ManyToOne

@JoinColumn**(**name **=** "course\_id"**)**

private Course course**;**

@Column**(**name **=** "registered\_at"**)**

private LocalDateTime registeredAt**;**

@Column**(**name **=** "grade"**)**

private int grade**;**

// additional properties

public CourseRegistration**()** **{**

**}**

public Long getId**()** **{**

**return** id**;**

**}**

public void setId**(**Long id**)** **{**

**this.**id **=** id**;**

**}**

public Student getStudent**()** **{**

**return** student**;**

**}**

public void setStudent**(**Student student**)** **{**

**this.**student **=** student**;**

**}**

public Course getCourse**()** **{**

**return** course**;**

**}**

public void setCourse**(**Course course**)** **{**

**this.**course **=** course**;**

**}**

public LocalDateTime getRegisteredAt**()** **{**

**return** registeredAt**;**

**}**

public void setRegisteredAt**(**LocalDateTime registeredAt**)** **{**

**this.**registeredAt **=** registeredAt**;**

**}**

public int getGrade**()** **{**

**return** grade**;**

**}**

public void setGrade**(**int grade**)** **{**

**this.**grade **=** grade**;**

**}**

@Override

public int hashCode**()** **{**

final int prime **=** 31**;**

int result **=** 1**;**

result **=** prime **\*** result **+** **((**id **==** **null)** **?** 0 **:** id**.**hashCode**());**

**return** result**;**

**}**

@Override

public boolean equals**(**Object obj**)** **{**

**if** **(this** **==** obj**)**

**return** **true;**

**if** **(**obj **==** **null)**

**return** **false;**

**if** **(**getClass**()** **!=** obj**.**getClass**())**

**return** **false;**

CourseRegistration other **=** **(**CourseRegistration**)** obj**;**

**if** **(**id **==** **null)** **{**

**if** **(**other**.**id **!=** **null)**

**return** **false;**

**}** **else** **if** **(!**id**.**equals**(**other**.**id**))**

**return** **false;**

**return** **true;**

**}**

**}**

----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**jpa**.**many\_to\_many**;**

**import** java**.**time**.**LocalDateTime**;**

**import** java**.**util**.**logging**.**Level**;**

**import** java**.**util**.**logging**.**Logger**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** ge**.**ufc**.**jpa**.**HibernateUtil**;**

public class Test **{**

private static SessionFactory sessionFactory**;**

private Session session**;**

static **{**

Logger**.**getLogger**(**"org.hibernate"**).**setLevel**(**Level**.**OFF**);**

sessionFactory **=** HibernateUtil**.**getSessionFactory**();**

**}**

**{**

session **=** sessionFactory**.**openSession**();**

session**.**beginTransaction**();**

**}**

public static void main**(**String**[]** args**)** **{**

Test test **=** **null;**

**try** **{**

test **=** **new** Test**();**

test**.**whenCourseLikePersisted\_thenCorrect**();**

test **=** **new** Test**();**

test**.**whenCourseRatingPersisted\_thenCorrect**();**

test **=** **new** Test**();**

test**.**whenCourseRegistrationPersisted\_thenCorrect**();**

**}** **finally** **{**

test**.**session**.**close**();**

sessionFactory**.**close**();**

**}**

**}**

private void whenCourseLikePersisted\_thenCorrect**()** **{**

Long studentId **=** 10L**;**

Long courseId **=** 20L**;**

Student student **=** **new** Student**(**studentId**);**

Course course **=** **new** Course**(**courseId**);**

student**.**getLikedCourses**().**add**(**course**);**

course**.**getLikes**().**add**(**student**);** // optional add, because Student is relationship owner

session**.**persist**(**student**);**

session**.**persist**(**course**);**

session**.**getTransaction**().**commit**();**

session**.**close**();**

session **=** sessionFactory**.**openSession**();**

Student persistedStudent **=** session**.**find**(**Student**.**class**,** studentId**);**

System**.**out**.**println**(**persistedStudent**.**getId**());** // studentId

System**.**out**.**println**(**persistedStudent**.**getLikedCourses**().**iterator**().**next**().**getId**());** // courseId

Course persistedCourse **=** session**.**find**(**Course**.**class**,** courseId**);**

System**.**out**.**println**(**persistedCourse**.**getId**());** // courseId

System**.**out**.**println**(**persistedCourse**.**getLikes**().**iterator**().**next**().**getId**());** // studentId

**}**

private void whenCourseRatingPersisted\_thenCorrect**()** **{**

Long studentId **=** 100L**;**

Long courseId **=** 200L**;**

Student student **=** **new** Student**(**studentId**);**

session**.**persist**(**student**);**

Course course **=** **new** Course**(**courseId**);**

session**.**persist**(**course**);**

CourseRating courseRating **=** **new** CourseRating**();**

courseRating**.**setId**(new** CourseRatingKey**());**

courseRating**.**setStudent**(**student**);**

courseRating**.**setCourse**(**course**);**

courseRating**.**setRating**(**77**);**

session**.**persist**(**courseRating**);**

session**.**getTransaction**().**commit**();**

session**.**close**();**

session **=** sessionFactory**.**openSession**();**

CourseRating persistedCourseRating **=** session**.**find**(**CourseRating**.**class**,** **new** CourseRatingKey**(**studentId**,** courseId**));**

System**.**out**.**println**(**persistedCourseRating**.**getStudent**().**getId**());** // studentId

System**.**out**.**println**(**persistedCourseRating**.**getCourse**().**getId**());** // courseId

System**.**out**.**println**(**courseRating**.**getRating**());** // 77

**}**

private void whenCourseRegistrationPersisted\_thenCorrect**()** **{**

Long studentId **=** 1000L**;**

Long courseId **=** 2000L**;**

Long courseRegistrationId **=** 1L**;**

Student student **=** **new** Student**(**studentId**);**

session**.**persist**(**student**);**

Course course **=** **new** Course**(**courseId**);**

session**.**persist**(**course**);**

CourseRegistration courseRegistration **=** **new** CourseRegistration**();**

courseRegistration**.**setId**(**courseRegistrationId**);**

courseRegistration**.**setStudent**(**student**);**

courseRegistration**.**setCourse**(**course**);**

courseRegistration**.**setRegisteredAt**(**LocalDateTime**.**now**());**

courseRegistration**.**setGrade**(**87**);**

session**.**persist**(**courseRegistration**);**

session**.**getTransaction**().**commit**();**

session**.**close**();**

session **=** sessionFactory**.**openSession**();**

CourseRegistration persistedCourseRegistration **=** session**.**find**(**CourseRegistration**.**class**,** courseRegistrationId**);**

System**.**out**.**println**(**persistedCourseRegistration**.**getId**());** // courseRegistrationId

System**.**out**.**println**(**persistedCourseRegistration**.**getStudent**().**getId**());** // studentId

System**.**out**.**println**(**persistedCourseRegistration**.**getCourse**().**getId**());** // courseId

System**.**out**.**println**(**persistedCourseRegistration**.**getRegisteredAt**());** // now

System**.**out**.**println**(**persistedCourseRegistration**.**getGrade**());** // 87

**}**

**}**

შემდეგ მაგალითში ვნახოთ, როგორ უნდა დავუკავშირდეთ ბაზას ვებ აპლიკაციიდან datasource და connection pooling-ის საშუალებით JPA-Hibernate გარემოში. გადავწეროთ წინა მაგალითებში გაკეთებულია MovieCrudService.

პირველ რიგში დავუკავშირდეთ derby-ს ბაზას კლიენტ/სერვერულ რეჟიმში, შევქმნათ აპლიკაციისთვის საჭირო ცხრილი და შევავსოთ სატესტო მონაცემებით.

ij> connect 'jdbc:derby://localhost:1527/sample';

**CREATE** **TABLE** Movies **(**

imdbId VARCHAR**(**50**),**

title VARCHAR**(**100**),**

**PRIMARY** **KEY** **(**imdbId**)**

**);**

**INSERT** **INTO** Movies

**(**imdbId**,** title**)**

**VALUES**

**(**'tt0096895'**,** 'Batman'**),**

**(**'tt0096900'**,** 'Into the Wild'**),**

**(**'tt0096950'**,** 'Gladiator'**);**

**SELECT** **\*** **FROM** Movies**;**

დავაგენერიროთ ახალი ვებ აპლიკაცია: mvn archetype:generate -DgroupId=ge.ufc.webapps -DartifactId=resteasy-web-service-db-jpa -Dversion=1.0 -DarchetypeGroupId=org.apache.maven.archetypes -DarchetypeArtifactId=maven-archetype-webapp -DarchetypeVersion=1.4 -DinteractiveMode=false

pom.xml

<project xmlns=**"http://maven.apache.org/POM/4.0.0"** xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"** xsi:schemaLocation=**"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"**>

<modelVersion>**4.0.0**</modelVersion>

<groupId>**ge.ufc.webapps**</groupId>

<artifactId>**resteasy-web-service-db-jpa**</artifactId>

<packaging>**war**</packaging>

<version>**1.0**</version>

<name>**resteasy-web-service-db-jpa**</name>

<url>**http://www.ufc.ge**</url>

<properties>

<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>

<maven.compiler.release>**11**</maven.compiler.release>

<jboss.resteasy.version>**3.15.3.Final**</jboss.resteasy.version>

<derby.version>**10.15.2.0**</derby.version>

<log4j2.version>**2.18.0**</log4j2.version>

<hibernate.version>**5.6.12.Final**</hibernate.version>

</properties>

<dependencies>

<dependency>

<groupId>**org.jboss.resteasy**</groupId>

<artifactId>**resteasy-jaxrs**</artifactId>

<version>**${jboss.resteasy.version}**</version>

</dependency>

<dependency>

<groupId>**org.jboss.resteasy**</groupId>

<artifactId>**resteasy-jackson-provider**</artifactId>

<version>**${jboss.resteasy.version}**</version>

</dependency>

<dependency>

<groupId>**org.jboss.resteasy**</groupId>

<artifactId>**resteasy-jaxb-provider**</artifactId>

<version>**${jboss.resteasy.version}**</version>

</dependency>

<dependency>

<groupId>**javax.xml.bind**</groupId>

<artifactId>**jaxb-api**</artifactId>

<version>**2.3.1**</version>

</dependency>

<dependency>

<groupId>**com.sun.xml.bind**</groupId>

<artifactId>**jaxb-core**</artifactId>

<version>**2.3.0.1**</version>

</dependency>

<dependency>

<groupId>**com.sun.xml.bind**</groupId>

<artifactId>**jaxb-impl**</artifactId>

<version>**2.3.2**</version>

</dependency>

<dependency>

<groupId>**org.apache.logging.log4j**</groupId>

<artifactId>**log4j-api**</artifactId>

<version>**${log4j2.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.apache.logging.log4j**</groupId>

<artifactId>**log4j-core**</artifactId>

<version>**${log4j2.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbyclient**</artifactId>

<version>**${derby.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbytools**</artifactId>

<version>**${derby.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbyshared**</artifactId>

<version>**${derby.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.hibernate**</groupId>

<artifactId>**hibernate-core**</artifactId>

<version>**${hibernate.version}**</version>

</dependency>

</dependencies>

<build>

<finalName>**resteasy-web-service-db-jpa**</finalName>

<plugins>

<plugin>

<artifactId>**maven-compiler-plugin**</artifactId>

<version>**3.8.1**</version>

</plugin>

<plugin>

<groupId>**org.apache.maven.plugins**</groupId>

<artifactId>**maven-eclipse-plugin**</artifactId>

<version>**2.10**</version>

<configuration>

<projectNameTemplate>

**[artifactId]-[version]**

</projectNameTemplate>

<wtpmanifest>**true**</wtpmanifest>

<wtpapplicationxml>**true**</wtpapplicationxml>

<wtpversion>**2.0**</wtpversion>

<manifest>

**${basedir}/src/main/resources/META-INF/MANIFEST.MF**

</manifest>

</configuration>

</plugin>

<plugin>

<groupId>**org.apache.tomcat.maven**</groupId>

<artifactId>**tomcat7-maven-plugin**</artifactId>

<version>**2.2**</version>

<configuration>

<url>**http://localhost:8080/manager/text**</url>

<server>**TomcatServer**</server>

<path>**/resteasy-web-service-db-jpa**</path>

</configuration>

</plugin>

</plugins>

</build>

</project>

აპლიკაცია უნდა დავა-deploy-ოთ პირველ რიგში tomcat-ზე, ამიტომ გვჭირდება tomcat7-maven-plugin.

web.xml

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd" >

<web-app>

<display-name>**RESTEasy Web Application**</display-name>

<context-param>

<param-name>**resteasy.resources**</param-name>

<param-value>**ge.ufc.webapps.ws.MovieCrudServiceImpl**</param-value>

</context-param>

<context-param>

<param-name>**resteasy.servlet.mapping.prefix**</param-name>

<param-value>**/**</param-value>

</context-param>

<listener>

<listener-class>**org.jboss.resteasy.plugins.server.servlet.ResteasyBootstrap**</listener-class>

</listener>

<servlet>

<servlet-name>**resteasy-servlet**</servlet-name>

<servlet-class>**org.jboss.resteasy.plugins.server.servlet.HttpServletDispatcher**</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>**resteasy-servlet**</servlet-name>

<url-pattern>**/\***</url-pattern>

</servlet-mapping>

</web-app>

src\main-ში შევქმნათ java ფოლდერი, mvn clean package გავბილდოთ პროექტი და შევაიმპორტოთ ეკლიპსში mvn eclipse:eclipse

პროექტს დავამატოთ კლასები შესაბამის პაკეტებში:

package ge**.**ufc**.**webapps**.**model**;**

**import** javax**.**persistence**.**Column**;**

**import** javax**.**persistence**.**Entity**;**

**import** javax**.**persistence**.**Id**;**

**import** javax**.**persistence**.**Table**;**

**import** javax**.**xml**.**bind**.**annotation**.**XmlAccessType**;**

**import** javax**.**xml**.**bind**.**annotation**.**XmlAccessorType**;**

**import** javax**.**xml**.**bind**.**annotation**.**XmlRootElement**;**

**import** javax**.**xml**.**bind**.**annotation**.**XmlType**;**

@XmlRootElement

@XmlAccessorType**(**XmlAccessType**.**FIELD**)**

@XmlType**(**name **=** "movie"**,** propOrder **=** **{** "imdbId"**,** "title" **})**

@Entity

@Table**(**name **=** "Movies"**)**

public class Movie **{**

@Id

@Column**(**name **=** "imdbId"**)**

private String imdbId**;**

@Column**(**name **=** "title"**)**

private String title**;**

public Movie**(**String imdbId**,** String title**)** **{**

**this.**imdbId **=** imdbId**;**

**this.**title **=** title**;**

**}**

public Movie**()** **{**

**}**

public String getImdbId**()** **{**

**return** imdbId**;**

**}**

public void setImdbId**(**String imdbId**)** **{**

**this.**imdbId **=** imdbId**;**

**}**

public String getTitle**()** **{**

**return** title**;**

**}**

public void setTitle**(**String title**)** **{**

**this.**title **=** title**;**

**}**

@Override

public String toString**()** **{**

StringBuilder sb **=** **new** StringBuilder**();**

sb**.**append**(**"{imdbId="**);**

sb**.**append**(**imdbId**);**

sb**.**append**(**", title="**);**

sb**.**append**(**title**);**

sb**.**append**(**"}"**);**

**return** sb**.**toString**();**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**model**;**

**import** java**.**util**.**List**;**

**import** javax**.**xml**.**bind**.**annotation**.**XmlAccessType**;**

**import** javax**.**xml**.**bind**.**annotation**.**XmlAccessorType**;**

**import** javax**.**xml**.**bind**.**annotation**.**XmlElement**;**

**import** javax**.**xml**.**bind**.**annotation**.**XmlRootElement**;**

**import** org**.**codehaus**.**jackson**.**annotate**.**JsonProperty**;**

@XmlRootElement**(**name **=** "movies"**)**

@XmlAccessorType**(**XmlAccessType**.**FIELD**)**

public class Movies **{**

@XmlElement**(**name **=** "movie"**)**

@JsonProperty**(**"movies"**)**

private List**<**Movie**>** movies**;**

public List**<**Movie**>** getMovies**()** **{**

**return** movies**;**

**}**

public void setMovies**(**List**<**Movie**>** movies**)** **{**

**this.**movies **=** movies**;**

**}**

@Override

public String toString**()** **{**

**return** movies**.**toString**();**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**model**;**

public class MovieNotFoundException **extends** Exception **{**

private static final long serialVersionUID **=** 1L**;**

public MovieNotFoundException**(**String message**)** **{**

**super(**message**);**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**model**;**

public class MovieAlreadyExistsException **extends** Exception **{**

private static final long serialVersionUID **=** 1L**;**

public MovieAlreadyExistsException**(**String message**)** **{**

**super(**message**);**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**dao**;**

public class DatabaseException **extends** Exception **{**

private static final long serialVersionUID **=** 1L**;**

public DatabaseException**(**String message**,** Throwable throwable**)** **{**

**super(**message**,** throwable**);**

**}**

public DatabaseException**(**String message**)** **{**

**super(**message**);**

**}**

public DatabaseException**(**Throwable throwable**)** **{**

**super(**throwable**);**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**dao**;**

**import** org**.**hibernate**.**SessionFactory**;**

**import** org**.**hibernate**.**boot**.**Metadata**;**

**import** org**.**hibernate**.**boot**.**MetadataSources**;**

**import** org**.**hibernate**.**boot**.**registry**.**StandardServiceRegistryBuilder**;**

**import** org**.**hibernate**.**service**.**ServiceRegistry**;**

public class HibernateUtil **{**

private static SessionFactory sessionFactory**;**

private HibernateUtil**()** **{**

**}**

public static SessionFactory getSessionFactory**()** **throws** DatabaseException **{**

**if** **(**sessionFactory **==** **null)** **{**

sessionFactory **=** buildSessionFactory**();**

**}**

**return** sessionFactory**;**

**}**

private static SessionFactory buildSessionFactory**()** **throws** DatabaseException **{**

**try** **{**

ServiceRegistry registry **=** **new** StandardServiceRegistryBuilder**().**configure**(**"hibernate.cfg.xml"**).**build**();**

MetadataSources metadataSources **=** **new** MetadataSources**(**registry**);**

Metadata metadata **=** metadataSources**.**getMetadataBuilder**().**build**();**

**return** metadata**.**getSessionFactoryBuilder**().**build**();**

**}** **catch** **(**Exception ex**)** **{**

**throw** **new** DatabaseException**(**"Unable to connect to the database"**,** ex**);**

**}**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**dao**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movie**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieAlreadyExistsException**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieNotFoundException**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movies**;**

public interface MoviesDao **{**

Movie getMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException**;**

void addMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieAlreadyExistsException**;**

void updateMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieNotFoundException**;**

void deleteMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException**;**

Movies listMovies**()** **throws** DatabaseException**;**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**dao**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**persistence**.**PersistenceException**;**

**import** javax**.**persistence**.**TypedQuery**;**

**import** org**.**hibernate**.**Session**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movie**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieAlreadyExistsException**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieNotFoundException**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movies**;**

public class MoviesDaoImpl\_Jpa **implements** MoviesDao **{**

@Override

public Movie getMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

Movie movie **=** session**.**find**(**Movie**.**class**,** imdbId**);**

**if** **(**movie **==** **null)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

**return** movie**;**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to get movie from the database"**,** e**);**

**}**

**}**

@Override

public void addMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieAlreadyExistsException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

**if** **(**session**.**find**(**Movie**.**class**,** movie**.**getImdbId**())** **!=** **null)** **{**

**throw** **new** MovieAlreadyExistsException**(**"Movie already exists in the database"**);**

**}**

session**.**getTransaction**().**begin**();**

session**.**persist**(**movie**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to add movie to the database"**,** e**);**

**}**

**}**

@Override

public void updateMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

**if** **(**session**.**find**(**Movie**.**class**,** movie**.**getImdbId**())** **==** **null)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

session**.**getTransaction**().**begin**();**

session**.**merge**(**movie**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to update movie to the database"**,** e**);**

**}**

**}**

@Override

public void deleteMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

Movie movie **=** session**.**find**(**Movie**.**class**,** imdbId**);**

**if** **(**movie **==** **null)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

session**.**getTransaction**().**begin**();**

session**.**remove**(**movie**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to delete movie from the database"**,** e**);**

**}**

**}**

@Override

public Movies listMovies**()** **throws** DatabaseException **{**

Movies movies **=** **new** Movies**();**

movies**.**setMovies**(new** ArrayList**<>());**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

TypedQuery**<**Movie**>** query **=** session**.**createQuery**(**"from Movie"**,** Movie**.**class**);**

movies**.**getMovies**().**addAll**(**query**.**getResultList**());**

**return** movies**;**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to get movies from the database"**,** e**);**

**}**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**dao**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**persistence**.**PersistenceException**;**

**import** javax**.**persistence**.**TypedQuery**;**

**import** org**.**hibernate**.**Session**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movie**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieAlreadyExistsException**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieNotFoundException**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movies**;**

public class MoviesDaoImpl\_Hibernate **implements** MoviesDao **{**

@Override

public Movie getMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

Movie movie **=** session**.**get**(**Movie**.**class**,** imdbId**);**

**if** **(**movie **==** **null)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

**return** movie**;**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to get movie from the database"**,** e**);**

**}**

**}**

@Override

public void addMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieAlreadyExistsException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

**if** **(**session**.**get**(**Movie**.**class**,** movie**.**getImdbId**())** **!=** **null)** **{**

**throw** **new** MovieAlreadyExistsException**(**"Movie already exists in the database"**);**

**}**

session**.**beginTransaction**();**

session**.**save**(**movie**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to add movie to the database"**,** e**);**

**}**

**}**

@Override

public void updateMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

Movie existingMovie **=** session**.**get**(**Movie**.**class**,** movie**.**getImdbId**());**

**if** **(**existingMovie **==** **null)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

session**.**evict**(**existingMovie**);**

session**.**beginTransaction**();**

session**.**update**(**movie**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to update movie to the database"**,** e**);**

**}**

**}**

@Override

public void deleteMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

Movie movie **=** session**.**get**(**Movie**.**class**,** imdbId**);**

**if** **(**movie **==** **null)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

session**.**beginTransaction**();**

session**.**delete**(**movie**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to delete movie from the database"**,** e**);**

**}**

**}**

@Override

public Movies listMovies**()** **throws** DatabaseException **{**

Movies movies **=** **new** Movies**();**

movies**.**setMovies**(new** ArrayList**<>());**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

TypedQuery**<**Movie**>** query **=** session**.**createQuery**(**"from Movie"**,** Movie**.**class**);**

movies**.**getMovies**().**addAll**(**query**.**getResultList**());**

**return** movies**;**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to get movies from the database"**,** e**);**

**}**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**dao**;**

**import** java**.**sql**.**SQLException**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**persistence**.**NoResultException**;**

**import** javax**.**persistence**.**PersistenceException**;**

**import** javax**.**persistence**.**Query**;**

**import** javax**.**persistence**.**TypedQuery**;**

**import** org**.**hibernate**.**Session**;**

**import** org**.**hibernate**.**exception**.**ConstraintViolationException**;**

**import** org**.**hibernate**.**query**.**NativeQuery**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movie**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieAlreadyExistsException**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieNotFoundException**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movies**;**

public class MoviesDaoImpl\_HQL **implements** MoviesDao **{**

private static final String DUPLICATE\_KEY\_ERROR **=** "23505"**;**

@Override

public Movie getMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

TypedQuery**<**Movie**>** query **=** session**.**createQuery**(**"select m from Movie m where m.imdbId = :imdbId"**,**

Movie**.**class**);**

query**.**setParameter**(**"imdbId"**,** imdbId**);**

**return** query**.**getSingleResult**();**

**}** **catch** **(**PersistenceException e**)** **{**

**if** **(**e **instanceof** NoResultException**)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

**throw** **new** DatabaseException**(**"Unable to get movie from the database"**,** e**);**

**}**

**}**

@Override

public void addMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieAlreadyExistsException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

NativeQuery**<?>** query **=** session**.**createNativeQuery**(**"INSERT INTO Movies (imdbId, title) VALUES (?, ?)"**);**

query**.**setParameter**(**1**,** movie**.**getImdbId**());**

query**.**setParameter**(**2**,** movie**.**getTitle**());**

session**.**beginTransaction**();**

query**.**executeUpdate**();**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

Throwable cause **=** e**.**getCause**();**

**if** **(**cause **!=** **null** **&&** cause **instanceof** ConstraintViolationException**)** **{**

SQLException ex **=** **((**ConstraintViolationException**)** cause**).**getSQLException**();**

**if** **(**DUPLICATE\_KEY\_ERROR**.**equals**(**ex**.**getSQLState**()))** **{**

**throw** **new** MovieAlreadyExistsException**(**"Movie already exists in the database"**);**

**}**

**}**

**throw** **new** DatabaseException**(**"Unable to add movie to the database"**,** e**);**

**}**

**}**

@Override

public void updateMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

Query query **=** session**.**createQuery**(**"update Movie m set m.title = :title where m.imdbId = :imdbId"**);**

query**.**setParameter**(**"title"**,** movie**.**getTitle**());**

query**.**setParameter**(**"imdbId"**,** movie**.**getImdbId**());**

session**.**beginTransaction**();**

int numRows **=** query**.**executeUpdate**();**

**if** **(**numRows **==** 0**)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to update movie to the database"**,** e**);**

**}**

**}**

@Override

public void deleteMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

Query query **=** session**.**createQuery**(**"delete from Movie m where m.imdbId = :imdbId"**);**

query**.**setParameter**(**"imdbId"**,** imdbId**);**

session**.**beginTransaction**();**

int numRows **=** query**.**executeUpdate**();**

**if** **(**numRows **==** 0**)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to delete movie from the database"**,** e**);**

**}**

**}**

@Override

public Movies listMovies**()** **throws** DatabaseException **{**

Movies movies **=** **new** Movies**();**

movies**.**setMovies**(new** ArrayList**<>());**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

TypedQuery**<**Movie**>** query **=** session**.**createQuery**(**"select m from Movie m"**,** Movie**.**class**);**

movies**.**getMovies**().**addAll**(**query**.**getResultList**());**

**return** movies**;**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to get movies from the database"**,** e**);**

**}**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**dao**;**

**import** java**.**util**.**ArrayList**;**

**import** javax**.**persistence**.**PersistenceException**;**

**import** javax**.**persistence**.**criteria**.**CriteriaBuilder**;**

**import** javax**.**persistence**.**criteria**.**CriteriaDelete**;**

**import** javax**.**persistence**.**criteria**.**CriteriaQuery**;**

**import** javax**.**persistence**.**criteria**.**CriteriaUpdate**;**

**import** javax**.**persistence**.**criteria**.**Predicate**;**

**import** javax**.**persistence**.**criteria**.**Root**;**

**import** org**.**hibernate**.**Session**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movie**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieAlreadyExistsException**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieNotFoundException**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movies**;**

public class MoviesDaoImpl\_Criteria **implements** MoviesDao **{**

@Override

public Movie getMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Movie**>** criteria **=** builder**.**createQuery**(**Movie**.**class**);**

Root**<**Movie**>** root **=** criteria**.**from**(**Movie**.**class**);**

Predicate predicate **=** builder**.**equal**(**root**.**get**(**"imdbId"**),** imdbId**);**

criteria**.**where**(**predicate**);**

Movie movie **=** session**.**createQuery**(**criteria**).**uniqueResult**();**

**if** **(**movie **==** **null)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

**return** movie**;**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to get movie from the database"**,** e**);**

**}**

**}**

@Override

public void addMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieAlreadyExistsException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

**if** **(**session**.**get**(**Movie**.**class**,** movie**.**getImdbId**())** **!=** **null)** **{**

**throw** **new** MovieAlreadyExistsException**(**"Movie already exists in the database"**);**

**}**

session**.**beginTransaction**();**

session**.**save**(**movie**);**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to add movie to the database"**,** e**);**

**}**

**}**

@Override

public void updateMovie**(**Movie movie**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaUpdate**<**Movie**>** criteria **=** builder**.**createCriteriaUpdate**(**Movie**.**class**);**

Root**<**Movie**>** root **=** criteria**.**from**(**Movie**.**class**);**

criteria**.**set**(**root**.**get**(**"title"**),** movie**.**getTitle**());**

Predicate predicate **=** builder**.**equal**(**root**.**get**(**"imdbId"**),** movie**.**getImdbId**());**

criteria**.**where**(**predicate**);**

session**.**beginTransaction**();**

int numRows **=** session**.**createQuery**(**criteria**).**executeUpdate**();**

**if** **(**numRows **==** 0**)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to update movie to the database"**,** e**);**

**}**

**}**

@Override

public void deleteMovie**(**String imdbId**)** **throws** DatabaseException**,** MovieNotFoundException **{**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaDelete**<**Movie**>** criteria **=** builder**.**createCriteriaDelete**(**Movie**.**class**);**

Root**<**Movie**>** root **=** criteria**.**from**(**Movie**.**class**);**

Predicate predicate **=** builder**.**equal**(**root**.**get**(**"imdbId"**),** imdbId**);**

criteria**.**where**(**predicate**);**

session**.**beginTransaction**();**

int numRows **=** session**.**createQuery**(**criteria**).**executeUpdate**();**

**if** **(**numRows **==** 0**)** **{**

**throw** **new** MovieNotFoundException**(**"Movie not found in the database"**);**

**}**

session**.**getTransaction**().**commit**();**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to delete movie from the database"**,** e**);**

**}**

**}**

@Override

public Movies listMovies**()** **throws** DatabaseException **{**

Movies movies **=** **new** Movies**();**

movies**.**setMovies**(new** ArrayList**<>());**

**try** **(**Session session **=** HibernateUtil**.**getSessionFactory**().**openSession**())** **{**

CriteriaBuilder builder **=** session**.**getCriteriaBuilder**();**

CriteriaQuery**<**Movie**>** criteria **=** builder**.**createQuery**(**Movie**.**class**);**

Root**<**Movie**>** root **=** criteria**.**from**(**Movie**.**class**);**

criteria**.**select**(**root**);**

movies**.**getMovies**().**addAll**(**session**.**createQuery**(**criteria**).**list**());**

**return** movies**;**

**}** **catch** **(**PersistenceException e**)** **{**

**throw** **new** DatabaseException**(**"Unable to get movies from the database"**,** e**);**

**}**

**}**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**ws**;**

**import** javax**.**ws**.**rs**.**Consumes**;**

**import** javax**.**ws**.**rs**.**DELETE**;**

**import** javax**.**ws**.**rs**.**GET**;**

**import** javax**.**ws**.**rs**.**POST**;**

**import** javax**.**ws**.**rs**.**PUT**;**

**import** javax**.**ws**.**rs**.**Path**;**

**import** javax**.**ws**.**rs**.**Produces**;**

**import** javax**.**ws**.**rs**.**QueryParam**;**

**import** javax**.**ws**.**rs**.**core**.**MediaType**;**

**import** javax**.**ws**.**rs**.**core**.**Response**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movie**;**

@Path**(**"/movies"**)**

public interface MovieCrudService **{**

@GET

@Path**(**"/getinfo"**)**

@Produces**({** MediaType**.**APPLICATION\_JSON**,** MediaType**.**APPLICATION\_XML **})**

Response movieByImdbId**(**@QueryParam**(**"imdbId"**)** String imdbId**);**

@POST

@Path**(**"/addmovie"**)**

@Consumes**({** MediaType**.**APPLICATION\_JSON**,** MediaType**.**APPLICATION\_XML **})**

Response addMovie**(**Movie movie**);**

@PUT

@Path**(**"/updatemovie"**)**

@Consumes**({** MediaType**.**APPLICATION\_JSON**,** MediaType**.**APPLICATION\_XML **})**

Response updateMovie**(**Movie movie**);**

@DELETE

@Path**(**"/deletemovie"**)**

Response deleteMovie**(**@QueryParam**(**"imdbId"**)** String imdbId**);**

@GET

@Path**(**"/listmovies"**)**

@Produces**({** MediaType**.**APPLICATION\_JSON**,** MediaType**.**APPLICATION\_XML **})**

Response listMovies**();**

**}**

-----------------------------------------------------------------------------------------------------------------------------------------

package ge**.**ufc**.**webapps**.**ws**;**

**import** javax**.**ws**.**rs**.**core**.**Response**;**

**import** javax**.**ws**.**rs**.**core**.**Response**.**Status**;**

**import** org**.**apache**.**logging**.**log4j**.**LogManager**;**

**import** org**.**apache**.**logging**.**log4j**.**Logger**;**

**import** ge**.**ufc**.**webapps**.**dao**.**DatabaseException**;**

**import** ge**.**ufc**.**webapps**.**dao**.**MoviesDao**;**

**import** ge**.**ufc**.**webapps**.**dao**.**MoviesDaoImpl\_Criteria**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movie**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieAlreadyExistsException**;**

**import** ge**.**ufc**.**webapps**.**model**.**MovieNotFoundException**;**

**import** ge**.**ufc**.**webapps**.**model**.**Movies**;**

public class MovieCrudServiceImpl **implements** MovieCrudService **{**

private static final Logger logger **=** LogManager**.**getLogger**();**

private static final MoviesDao moviesDao **=** **new** MoviesDaoImpl\_Criteria**();**

@Override

public Response movieByImdbId**(**String imdbId**)** **{**

**try** **{**

logger**.**info**(**"Getting movie from the database, imdbId: {}"**,** imdbId**);**

Movie movie **=** moviesDao**.**getMovie**(**imdbId**);**

logger**.**trace**(**"Movie got from the database: {}"**,** movie**);**

**return** Response**.**status**(**Status**.**OK**).**entity**(**movie**).**build**();**

**}** **catch** **(**MovieNotFoundException e**)** **{**

logger**.**warn**(**e**.**getMessage**());**

**return** Response**.**status**(**Status**.**NOT\_FOUND**).**build**();**

**}** **catch** **(**DatabaseException e**)** **{**

logger**.**error**(**"Database exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}** **catch** **(**Exception e**)** **{**

logger**.**error**(**"Unknown exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}**

**}**

@Override

public Response addMovie**(**Movie movie**)** **{**

**try** **{**

logger**.**info**(**"Adding movie to the database: {}"**,** movie**);**

moviesDao**.**addMovie**(**movie**);**

logger**.**trace**(**"Movie added to the database"**);**

**return** Response**.**status**(**Status**.**CREATED**).**build**();**

**}** **catch** **(**MovieAlreadyExistsException e**)** **{**

logger**.**warn**(**e**.**getMessage**());**

**return** Response**.**status**(**Status**.**CONFLICT**).**build**();**

**}** **catch** **(**DatabaseException e**)** **{**

logger**.**error**(**"Database exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}** **catch** **(**Exception e**)** **{**

logger**.**error**(**"Unknown exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}**

**}**

@Override

public Response updateMovie**(**Movie movie**)** **{**

**try** **{**

logger**.**info**(**"Updating movie to the database: {}"**,** movie**);**

moviesDao**.**updateMovie**(**movie**);**

logger**.**trace**(**"Movie updated to the database"**);**

**return** Response**.**status**(**Status**.**OK**).**build**();**

**}** **catch** **(**MovieNotFoundException e**)** **{**

logger**.**warn**(**e**.**getMessage**());**

**return** Response**.**status**(**Status**.**NOT\_FOUND**).**build**();**

**}** **catch** **(**DatabaseException e**)** **{**

logger**.**error**(**"Database exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}** **catch** **(**Exception e**)** **{**

logger**.**error**(**"Unknown exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}**

**}**

@Override

public Response deleteMovie**(**String imdbId**)** **{**

**try** **{**

logger**.**info**(**"Deleting movie from the database, imdbId: {}"**,** imdbId**);**

moviesDao**.**deleteMovie**(**imdbId**);**

logger**.**trace**(**"Movie deleted from the database"**);**

**return** Response**.**status**(**Status**.**OK**).**build**();**

**}** **catch** **(**MovieNotFoundException e**)** **{**

logger**.**warn**(**e**.**getMessage**());**

**return** Response**.**status**(**Status**.**NOT\_FOUND**).**build**();**

**}** **catch** **(**DatabaseException e**)** **{**

logger**.**error**(**"Database exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}** **catch** **(**Exception e**)** **{**

logger**.**error**(**"Unknown exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}**

**}**

@Override

public Response listMovies**()** **{**

**try** **{**

logger**.**info**(**"Getting movies from the database"**);**

Movies movies **=** moviesDao**.**listMovies**();**

logger**.**trace**(**"Movies got from the database: {}"**,** movies**);**

**return** Response**.**status**(**Status**.**OK**).**entity**(**movies**).**build**();**

**}** **catch** **(**DatabaseException e**)** **{**

logger**.**error**(**"Database exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}** **catch** **(**Exception e**)** **{**

logger**.**error**(**"Unknown exception"**,** e**);**

**return** Response**.**status**(**Status**.**INTERNAL\_SERVER\_ERROR**).**build**();**

**}**

**}**

**}**

პროექტში დამატებულია MoviesDao ინტერფეისის 4 სხვადასხვა რეალიზაცია:

1. MoviesDaoImpl\_Jpa - ბაზის CRUD (Create\_Read\_Update\_Delete) ოპერაციები რეალიზებულია jpa-ს სტანდარტიზებული მეთოდებით.
2. MoviesDaoImpl\_Hibernate - ბაზის CRUD (Create\_Read\_Update\_Delete) ოპერაციები რეალიზებულია hibernate-ის დამატებითი მეთოდებით.
3. MoviesDaoImpl\_HQL - ბაზის CRUD (Create\_Read\_Update\_Delete) ოპერაციები რეალიზებულია HQL ბრძანებებით.
4. MoviesDaoImpl\_Criteria - ბაზის CRUD (Create\_Read\_Update\_Delete) ოპერაციები რეალიზებულია Creiteria API-ით.

MovieCrudServiceImpl შეგვიძლია გამოვიყენოთ ამ ოთხიდან ნებისმიერი ერთ-ერთი რეალიზაცია: private static final MoviesDao moviesDao = new MoviesDaoImpl\_Criteria();

პირველად ვებ აპლიკაცია დავა-deploy-ოთ ტომკეტზე. გავთიშოთ ტომკეტი, თუ ჩართულია, და მანამდე ჩავატაროთ მოსამზადებელი სამუშაოები.

(თუ მანამდე არ გვაქვს გაკეთებული) ტომკეტის lib საქაღალდეში ჩავამატოთ: derbyclient-10.15.2.0.jar, derbyshared-10.15.2.0.jar და derbytools-10.15.2.0.jar ფაილები. ასევე აქვე უნდა გვქონდეს log4j-api-2.18.0.jar და log4j-core-2.18.0.jar ფაილები.

შემდეგ გადავიდეთ ტომკეტის conf დირექტორიაზე. log4j2\log4j2.xml ფაილი გვაქვს ასეთი შიგთავსით და log4j2 დირექტორია დამატებული უნდა გვქონდეს common.loader-ში:

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<Configuration>

<Appenders>

<File name=**"MyFile"** fileName=**"${sys:catalina.home}/logs/app.log"**>

<PatternLayout>

<Pattern>**%d{yyyy-MM-dd HH:mm:ss} %-5p %c %m%n**</Pattern>

</PatternLayout>

</File>

</Appenders>

<Loggers>

<Logger name=**"ge.ufc"** level=**"TRACE"** additivity = **"false"**>

<AppenderRef ref=**"MyFile"**/>

</Logger>

</Loggers>

</Configuration>

(თუ მანამდე არ გვაქვს გაკეთებული) ტომკეტი გაშვებისას ბაზის jar ფაილებზე catalina-ს ლოგში წერს warn-ების exception-ებს. ამის გათიშვა შეგვიძლია ტომკეტის conf-ის context.xml ფაილის ბოლოში, </Context> ტეგის დახურვამდე, <JarScanner scanManifest="false"/> ჩამატებით.

შემდეგ ჯერზე უნდა შევქმნათ Hibernate-ის კონფიგურაციის ფაილი. conf დირექტორიაში შევქმნათ resteasy-web-service-db-jpa საქაღალდე და catalina.properties-ის common.loader-ში დავამატოთ: "${catalina.home}/conf/resteasy-web-service-db-jpa"

resteasy-web-service-db-jpa საქაღალდეში დავამატოთ ფაილი hibernate.cfg.xml შემდეგი შიგთავსით:

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=**"hibernate.connection.datasource"**>**java:comp/env/jdbc/movieDS\_JPA**</property>

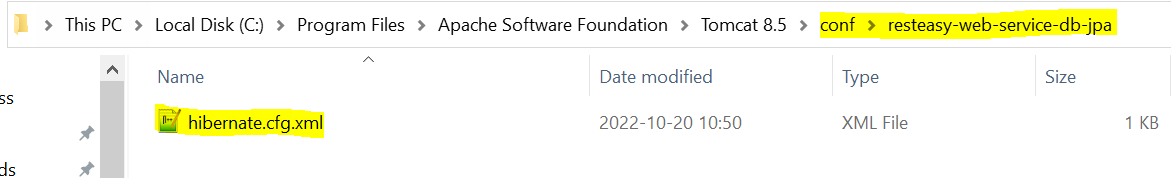
<property name=**"hibernate.dialect"**>**org.hibernate.dialect.DerbyTenSevenDialect**</property>

<property name=**"hibernate.show\_sql"**>**true**</property>

<mapping class=**"ge.ufc.webapps.model.Movie"**></mapping>

</session-factory>

</hibernate-configuration>



დაგვრჩა ბაზასთან დასაკავშირებელი დატასორს ფაილის შექმნა. ტომკეტის conf-ის Catalina\localhost დირექტორიებში (თუ არ არსებობს, ხელით უნდა შექმნათ ეს დირექტორიები. common.loader-ში დამატებას ეს დირექტორიები არ საჭიროებენ) უნდა ჩავამატოთ xml ფაილი, რომელსაც ერქმევა აპლიკაციის war ფაილის იდენტური სახელი, ანუ ამ შემთხვევაში: resteasy-web-service-db-jpa.xml

<Context>

<Resource

name=**"jdbc/movieDS\_JPA"**

auth=**"Container"**

factory=**"org.apache.tomcat.dbcp.dbcp2.BasicDataSourceFactory"**

type=**"javax.sql.DataSource"**

driverClassName=**"org.apache.derby.jdbc.ClientDriver"**

url=**"jdbc:derby://localhost:1527/sample"**

username=**"APP"**

password=**"123"**

defaultQueryTimeout=**"150"**

maxTotal=**"30"**

maxIdle=**"10"**

maxWaitMillis=**"1000"**

validationQuery=**"VALUES(1)"**

validationQueryTimeout=**"5"**

testWhileIdle=**"true"**

timeBetweenEvictionRunsMillis=**"5000"**

accessToUnderlyingConnectionAllowed=**"true"**

removeAbandonedOnMaintenance=**"true"**

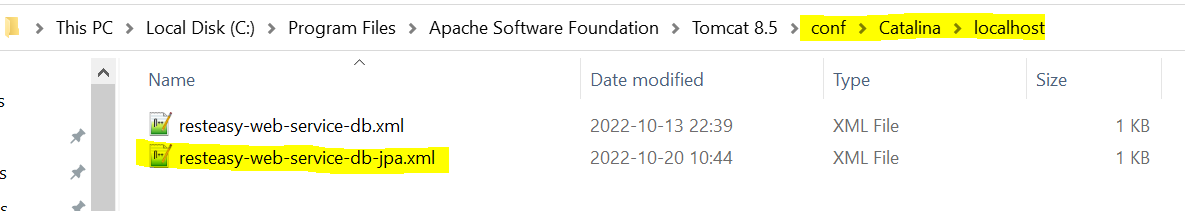
removeAbandonedOnBorrow=**"true"**

removeAbandonedTimeout=**"180"**

logAbandoned=**"true"**

/>

</Context>



გავბილდოთ პროექტი mvn clean package და მიღებული resteasy-web-service-db-jpa.war ფაილი ჩავაგდოთ ტომკეტის webapps-ში.

ჩავრთოთ ტომკეტი და დავიწყოთ ტესტირება. ყოველი მეთოდის გამოძახებაზე ასევე უნდა ნახოთ app.log ფაილში დალოგილი ინფორმაცია:

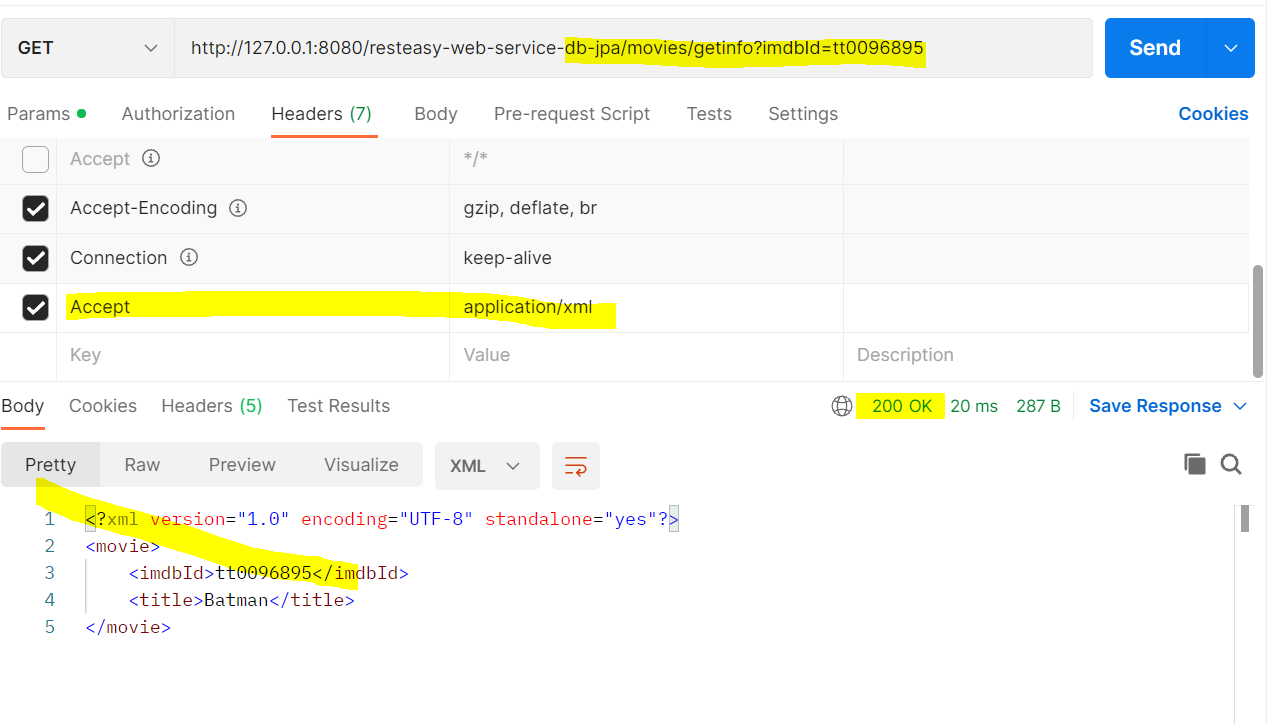
HTTP GET

http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/getinfo?imdbId=tt0096895

http\_request\_headers{

Accept: application/xml

}



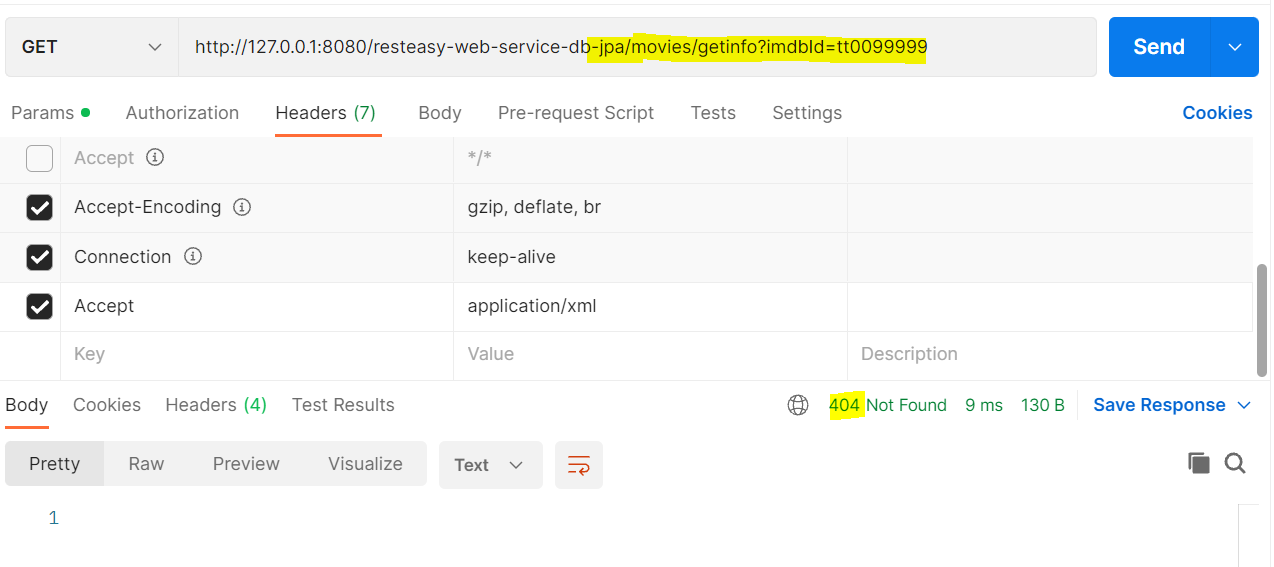
HTTP GET

http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/getinfo?imdbId=tt0099999

http\_request\_headers{

Accept: application/xml

}



HTTP POST

http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/addmovie

http\_request\_headers{

Content-Type: application/json

}

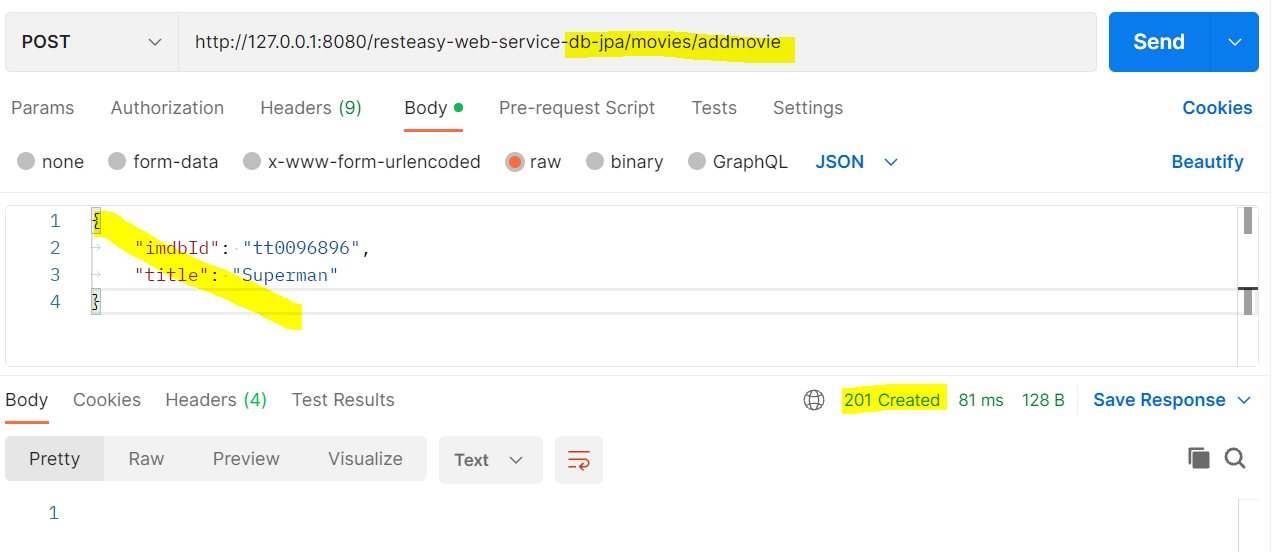
POST DATA:

{

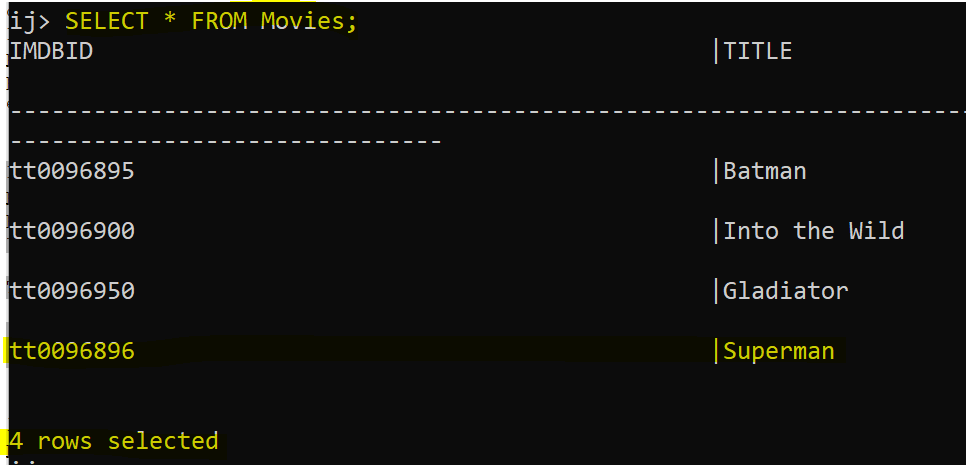
"imdbId": "tt0096896",

"title": "Superman"

}



SELECT \* FROM Movies;



HTTP POST

http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/addmovie

http\_request\_headers{

Content-Type: application/json

}

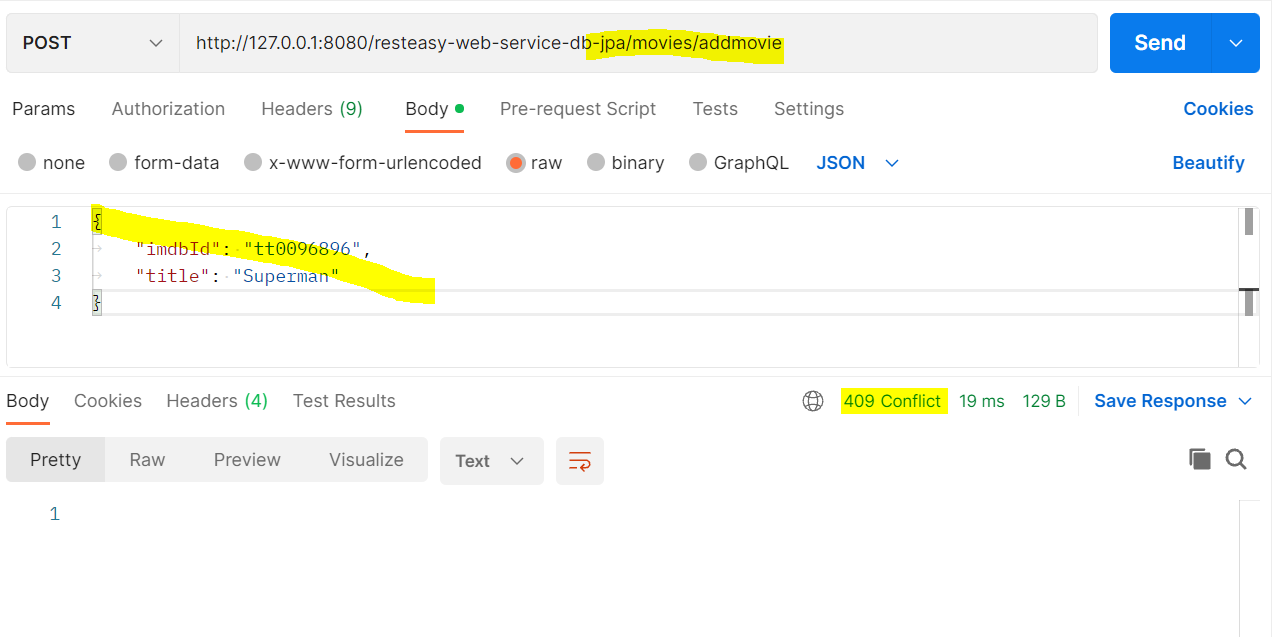
POST DATA:

{

"imdbId": "tt0096896",

"title": "Superman"

}



HTTP PUT

http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/updatemovie

http\_request\_headers{

Content-Type: application/xml

}

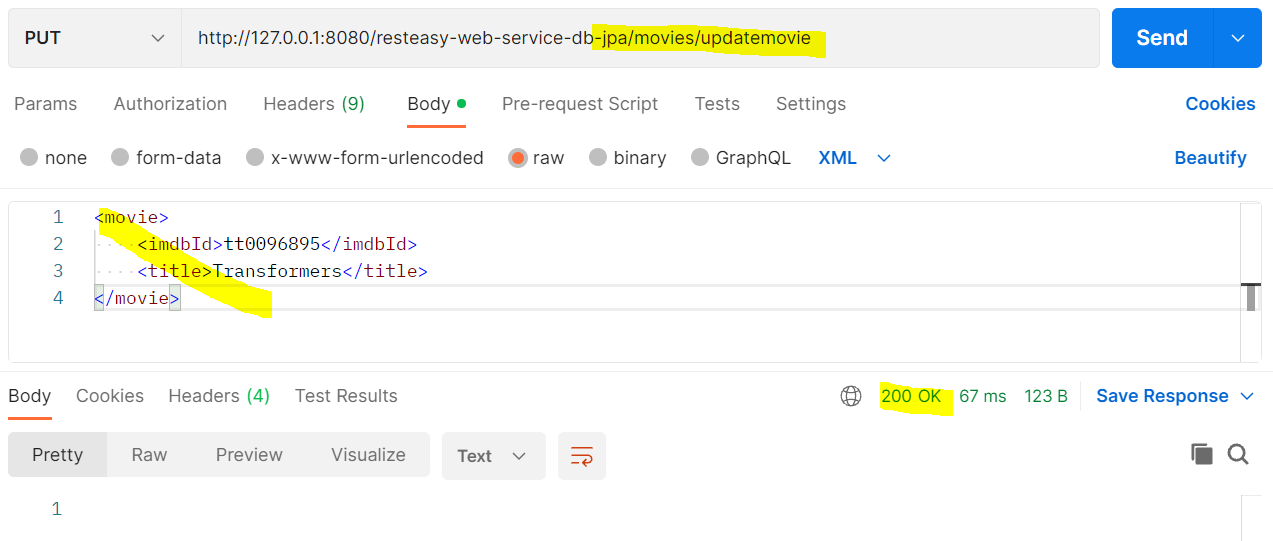
PUT DATA:

<movie>

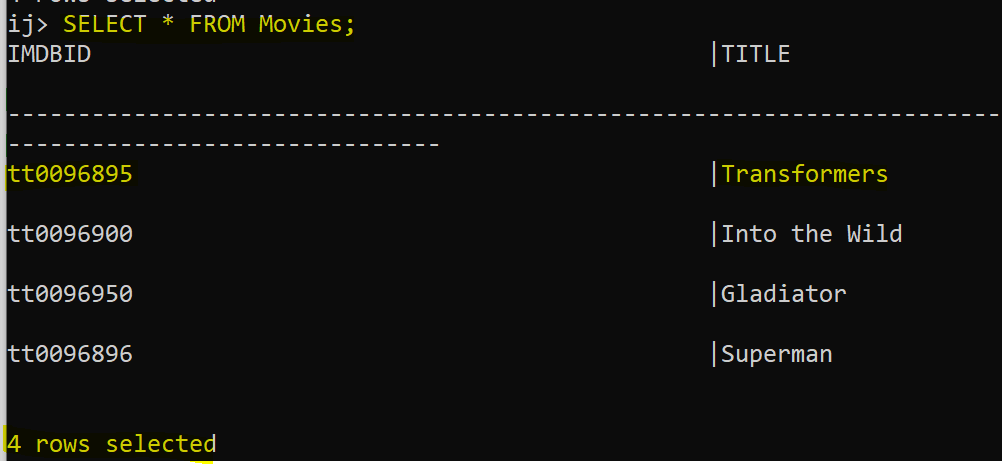
<imdbId>tt0096895</imdbId>

<title>Transformers</title>

</movie>



SELECT \* FROM Movies;



HTTP PUT

http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/updatemovie

http\_request\_headers{

Content-Type: application/xml

}

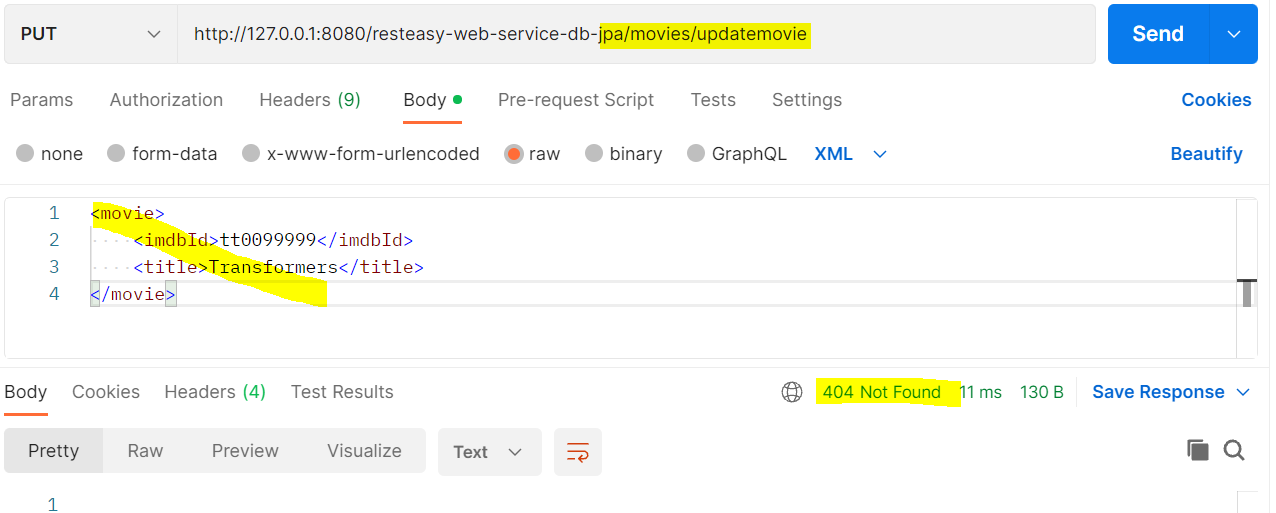
PUT DATA:

<movie>

<imdbId>tt0099999</imdbId>

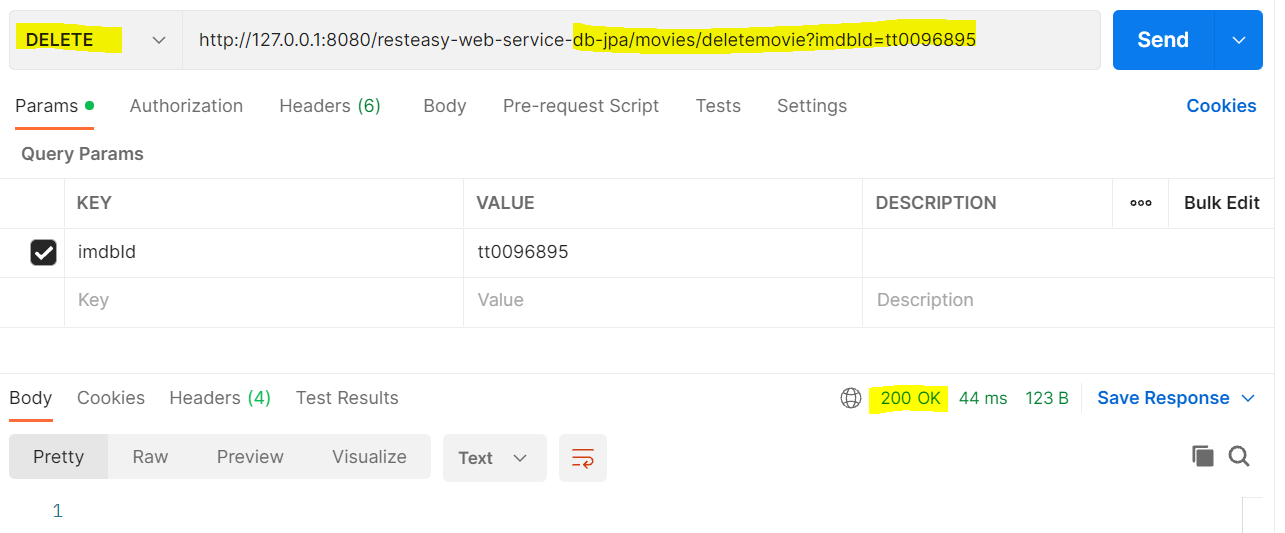
<title>Transformers</title>

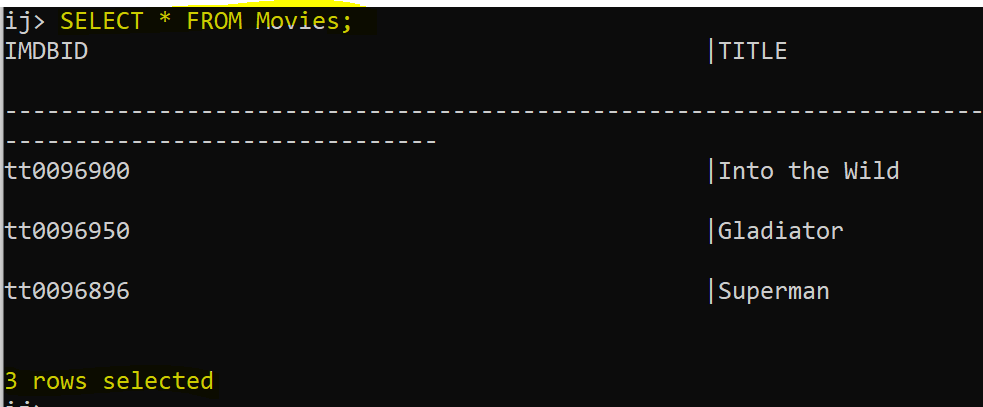
</movie>



HTTP DELETE

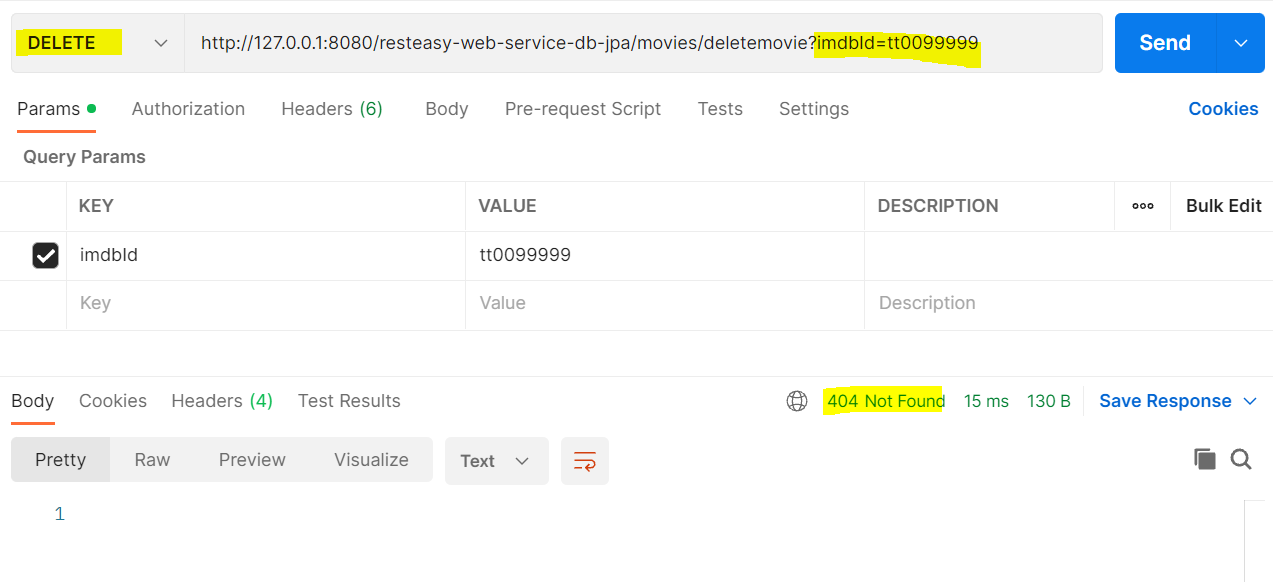
<http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/deletemovie?imdbId=tt0096895>





HTTP DELETE

<http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/deletemovie?imdbId=tt0099999>



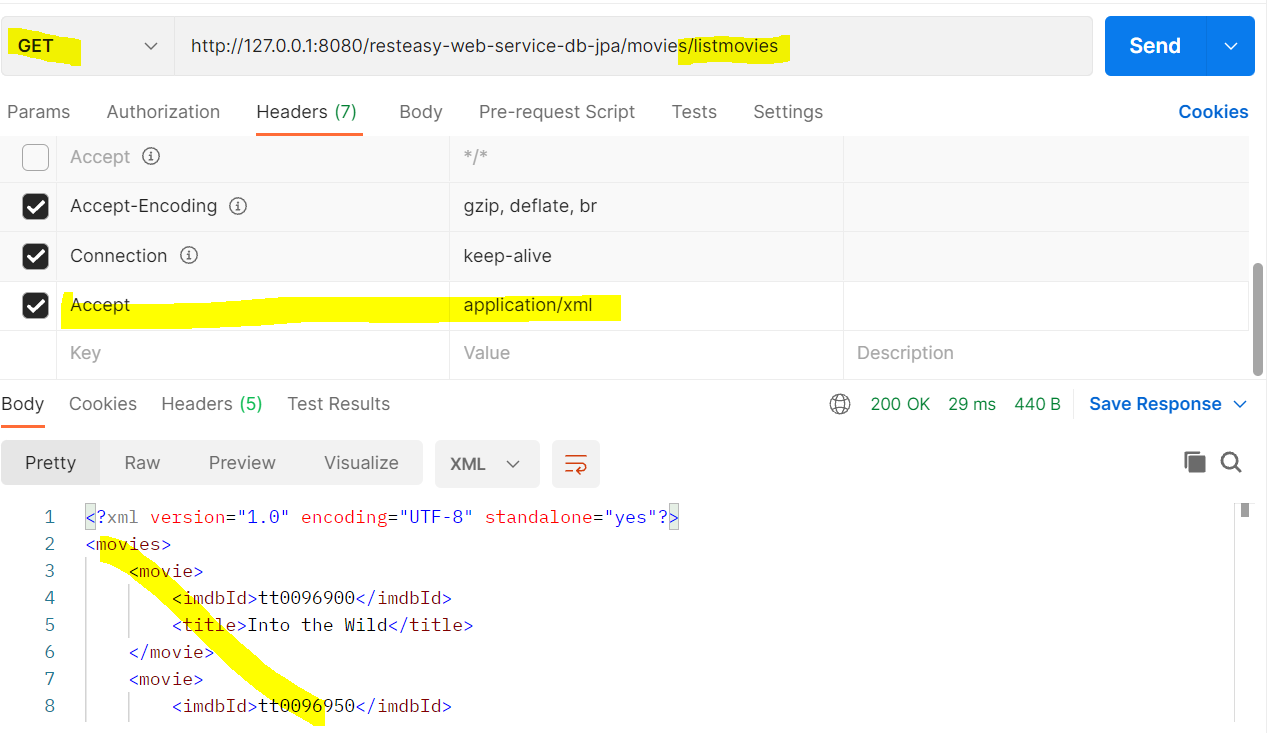
HTTP GET

http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/listmovies

http\_request\_headers{

Accept: application/xml

}

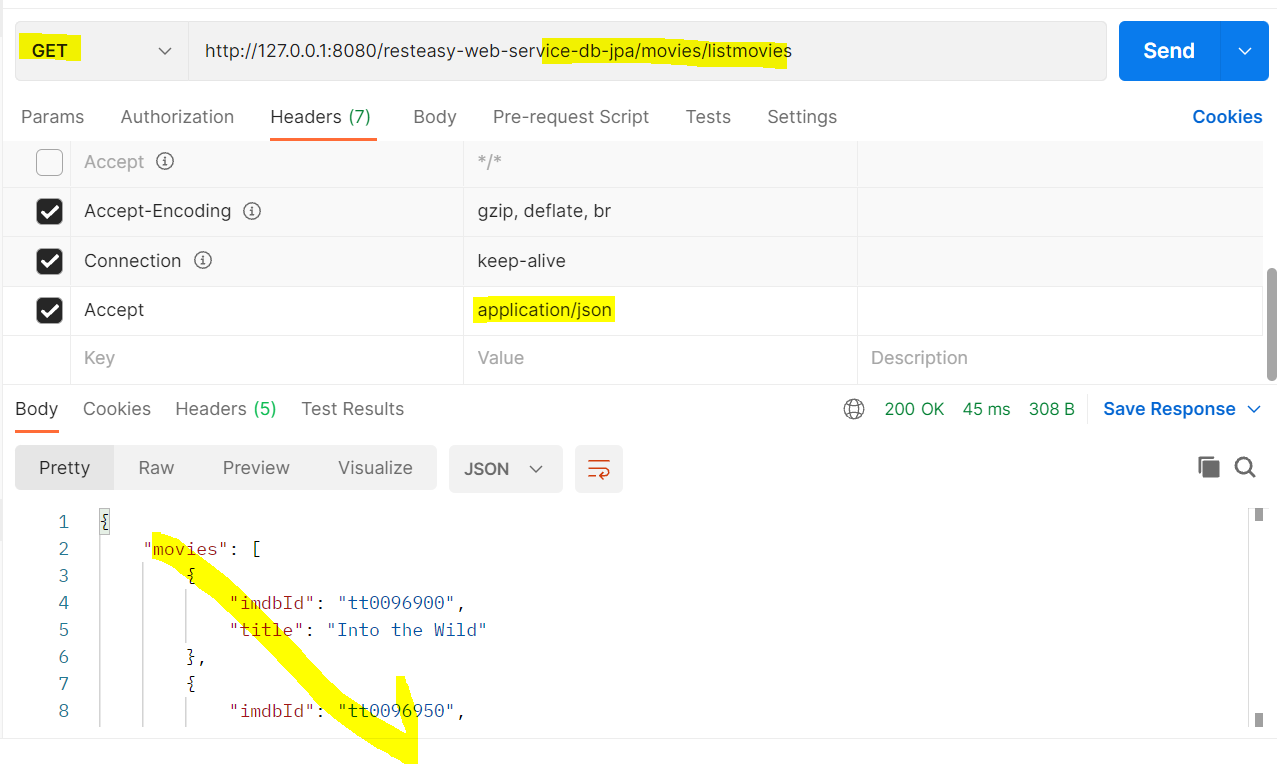


http://127.0.0.1:8080/resteasy-web-service-db-jpa/movies/listmovies

http\_request\_headers{

Accept: application/json

}



ახლა კი გადავიდეთ Wildfly-ზე. პირველ რიგში, გავთიშოთ იგი, თუ ჩართულია. პროექტი გადავაკოპიროთ resteasy-web-service-db-jpa-wildfly დირქტორიაში და pom.xml გადავაკეთოთ ასე:

<project xmlns=**"http://maven.apache.org/POM/4.0.0"** xmlns:xsi=**"http://www.w3.org/2001/XMLSchema-instance"** xsi:schemaLocation=**"http://maven.apache.org/POM/4.0.0 http://maven.apache.org/maven-v4\_0\_0.xsd"**>

<modelVersion>**4.0.0**</modelVersion>

<groupId>**ge.ufc.webapps**</groupId>

<artifactId>**resteasy-web-service-db-jpa-wildfly**</artifactId>

<packaging>**war**</packaging>

<version>**1.0**</version>

<name>**resteasy-web-service-db-jpa-wildfly**</name>

<url>**http://www.ufc.ge**</url>

<properties>

<project.build.sourceEncoding>**UTF-8**</project.build.sourceEncoding>

<maven.compiler.release>**11**</maven.compiler.release>

<jboss.resteasy.version>**3.15.3.Final**</jboss.resteasy.version>

<derby.version>**10.15.2.0**</derby.version>

<log4j2.version>**2.18.0**</log4j2.version>

<hibernate.version>**5.3.10.Final**</hibernate.version>

</properties>

<dependencies>

<dependency>

<groupId>**org.jboss.resteasy**</groupId>

<artifactId>**resteasy-jaxrs**</artifactId>

<version>**${jboss.resteasy.version}**</version>

</dependency>

<dependency>

<groupId>**org.jboss.resteasy**</groupId>

<artifactId>**resteasy-jackson-provider**</artifactId>

<version>**${jboss.resteasy.version}**</version>

</dependency>

<dependency>

<groupId>**org.jboss.resteasy**</groupId>

<artifactId>**resteasy-jaxb-provider**</artifactId>

<version>**${jboss.resteasy.version}**</version>

</dependency>

<dependency>

<groupId>**javax.xml.bind**</groupId>

<artifactId>**jaxb-api**</artifactId>

<version>**2.3.1**</version>

</dependency>

<dependency>

<groupId>**com.sun.xml.bind**</groupId>

<artifactId>**jaxb-core**</artifactId>

<version>**2.3.0.1**</version>

</dependency>

<dependency>

<groupId>**com.sun.xml.bind**</groupId>

<artifactId>**jaxb-impl**</artifactId>

<version>**2.3.2**</version>

</dependency>

<dependency>

<groupId>**org.apache.logging.log4j**</groupId>

<artifactId>**log4j-api**</artifactId>

<version>**${log4j2.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.apache.logging.log4j**</groupId>

<artifactId>**log4j-core**</artifactId>

<version>**${log4j2.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbyclient**</artifactId>

<version>**${derby.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbytools**</artifactId>

<version>**${derby.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.apache.derby**</groupId>

<artifactId>**derbyshared**</artifactId>

<version>**${derby.version}**</version>

<scope>**provided**</scope>

</dependency>

<dependency>

<groupId>**org.hibernate**</groupId>

<artifactId>**hibernate-core**</artifactId>

<version>**${hibernate.version}**</version>

<scope>**provided**</scope>

</dependency>

</dependencies>

<build>

<finalName>**resteasy-web-service-db-jpa-wildfly**</finalName>

<plugins>

<plugin>

<artifactId>**maven-compiler-plugin**</artifactId>

<version>**3.8.1**</version>

</plugin>

<plugin>

<groupId>**org.apache.maven.plugins**</groupId>

<artifactId>**maven-eclipse-plugin**</artifactId>

<version>**2.10**</version>

<configuration>

<projectNameTemplate>

**[artifactId]-[version]**

</projectNameTemplate>

<wtpmanifest>**true**</wtpmanifest>

<wtpapplicationxml>**true**</wtpapplicationxml>

<wtpversion>**2.0**</wtpversion>

<manifest>

**${basedir}/src/main/resources/META-INF/MANIFEST.MF**

</manifest>

</configuration>

</plugin>

<plugin>

<groupId>**org.wildfly.plugins**</groupId>

<artifactId>**wildfly-maven-plugin**</artifactId>

<version>**3.0.2.Final**</version>

</plugin>

<plugin>

<groupId>**org.apache.maven.plugins**</groupId>

<artifactId>**maven-war-plugin**</artifactId>

<version>**3.3.2**</version>

<configuration>

<archive>

<manifest>

<addDefaultImplementationEntries>**true**</addDefaultImplementationEntries>

</manifest>

<manifestEntries>

<Implementation-Build>**${buildNumber}**</Implementation-Build>

<Dependencies>**ge.ufc.webapps, org.apache.log4j2, org.apache.derby, org.hibernate**</Dependencies>

</manifestEntries>

</archive>

</configuration>

</plugin>

</plugins>

</build>

</project>

Wildfly-ს მოჰყვება hibernate-ის ბიბლიოთეკები და ამიტომ შესაბამისი ვერსიის dependency გაწერილია <scope>provided</scope>. დავამატეთ wildfly-maven-plugin და maven-war-plugin შესაბამისი Dependencies-ებით. აქ სიაში დაემატა: org.hibernate

წინა მაგალითებში მოდულებში უკვე დავაინსტალირეთ log4j2-ის ბიბლიოთეკები და ასევე დავამატეთ ge.ufc.webapps მოდული, სადაც გვაქვს შენახული log4j2.xml ფაილი:

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<Configuration>

<Appenders>

<File name=**"MyFile"** fileName=**"${sys:jboss.server.log.dir}/app.log"**>

<PatternLayout>

<Pattern>**%d{yyyy-MM-dd HH:mm:ss} %-5p %c %m%n**</Pattern>

</PatternLayout>

</File>

</Appenders>

<Loggers>

<Logger name=**"ge.ufc"** level=**"TRACE"** additivity = **"false"**>

<AppenderRef ref=**"MyFile"**/>

</Logger>

</Loggers>

</Configuration>

ge.ufc.webapps მოდულში, modules\system\layers\base\ge\ufc\webapps\main დირექტორიაში დავამატოთ ფაილი hibernate.cfg.xml

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=**"hibernate.connection.datasource"**>**java:jboss/datasources/movieDS\_JPA**</property>

<property name=**"hibernate.dialect"**>**org.hibernate.dialect.DerbyTenSevenDialect**</property>

<property name=**"hibernate.show\_sql"**>**true**</property>

<mapping class=**"ge.ufc.webapps.model.Movie"**></mapping>

</session-factory>

</hibernate-configuration>

(თუ მანამდე არ გვაქვს გაკეთებული) დავაინსტალიროთ ახალი მოდული დერბისთვის. modules\system\layers\base\org\apache დირექტორიაში შევქმნათ derby\main დირექტორიები და მასში ჩავყაროთ derbyclient-10.15.2.0.jar, derbyshared-10.15.2.0.jar და derbytools-10.15.2.0.jar ფაილები. აქვე შევქმნათ module.xml ფაილი შემდეგი შიგთავსით:

<?xml version=**"1.0"** encoding=**"UTF-8"**?>

<module name=**"org.apache.derby"** xmlns=**"urn:jboss:module:1.5"**>

<dependencies>

<module name=**"javax.api"**/>

<module name=**"javax.transaction.api"**/>

<module name=**"javax.servlet.api"** optional=**"true"**/>

</dependencies>

<resources>

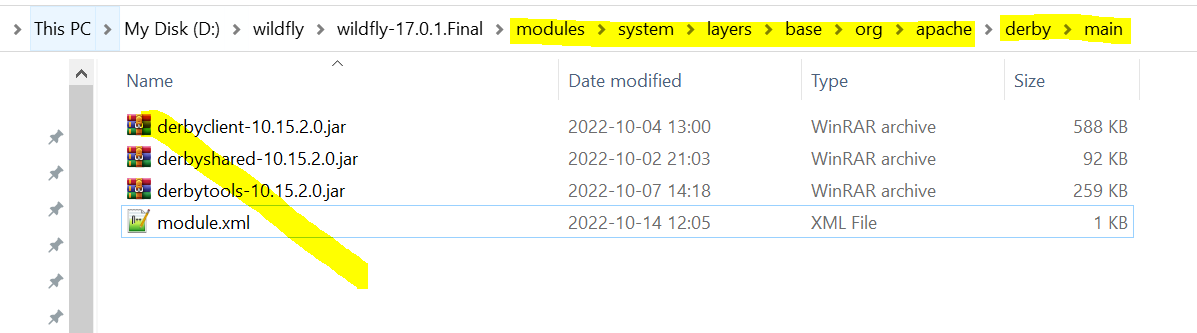
<resource-root path=**"derbyclient-10.15.2.0.jar"**/>

<resource-root path=**"derbyshared-10.15.2.0.jar"**/>

<resource-root path=**"derbytools-10.15.2.0.jar"**/>

</resources>

</module>



ამის შემდეგ driver (თუ მანამდე არ გვაქვს გაკეთებული) და datasource უნდა დავამატოთ standalone\configuration\standalone.xml ფაილში. მაგალითსვთის აქ უკვე დამატებულია h2 in-memory ბაზის driver და datasource. <datasources> სექციის <drivers> სექციაში დავამატოთ:

<driver name=**"derby"** module=**"org.apache.derby"**>

<xa-datasource-class>**org.apache.derby.jdbc.ClientXADataSource**</xa-datasource-class>

</driver>

დავამატოთ ასევე <datasource> ელემენტი:

<datasource jndi-name=**"java:jboss/datasources/movieDS\_JPA"** pool-name=**"movieDS\_JPA"** enabled=**"true"** use-java-context=**"true"**>

<connection-url>**jdbc:derby://localhost:1527/sample**</connection-url>

<driver>**derby**</driver>

<pool>

<min-pool-size>**5**</min-pool-size>

<max-pool-size>**20**</max-pool-size>

</pool>

<security>

<user-name>**APP**</user-name>

<password>**123**</password>

</security>

<validation>

<check-valid-connection-sql>**VALUES(1)**</check-valid-connection-sql>

</validation>

<timeout>

<blocking-timeout-millis>**60000**</blocking-timeout-millis>

<idle-timeout-minutes>**15**</idle-timeout-minutes>

</timeout>

<statement>

<track-statements>**true**</track-statements>

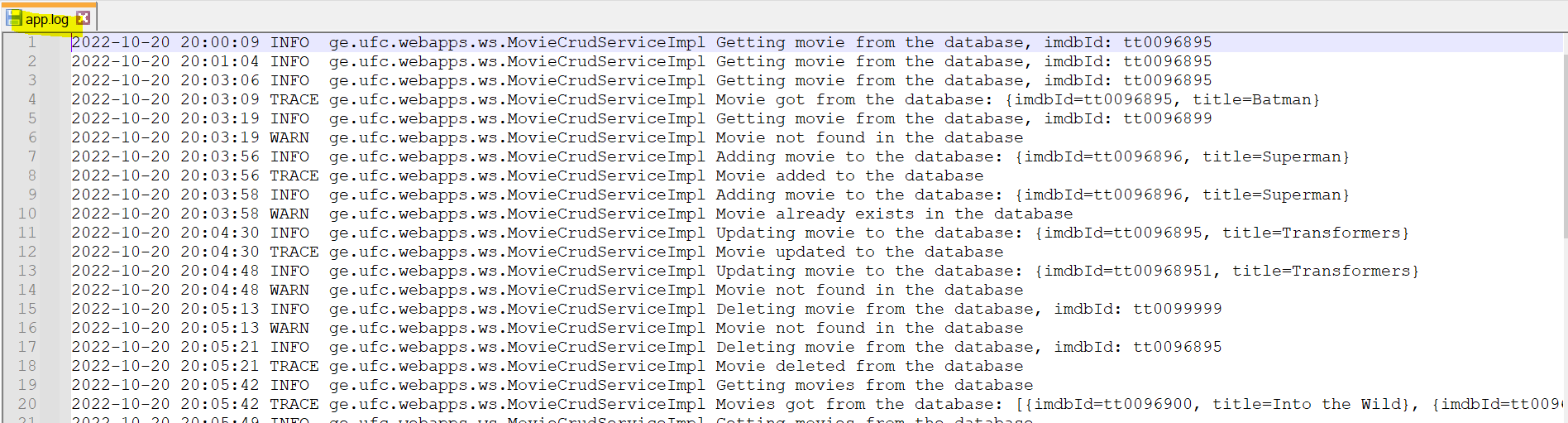
<prepared-statement-cache-size>**100**</prepared-statement-cache-size>

</statement>

</datasource>

როგორც ვხედავთ, wildfly-ს შემთხვავში jndi-ის name გამოიყურება ასე: java:jboss/datasources/movieDS\_JPA

ჩავრთოთ wildfly და დავა-deploy-ოთ mvn wildfly:deploy და გავაკეთოთ ვებ სერვისის რამდენიმე სატესტო გამოძახება და ასევე დავაკვირდეთ standalone\log\app.log ფაილს.



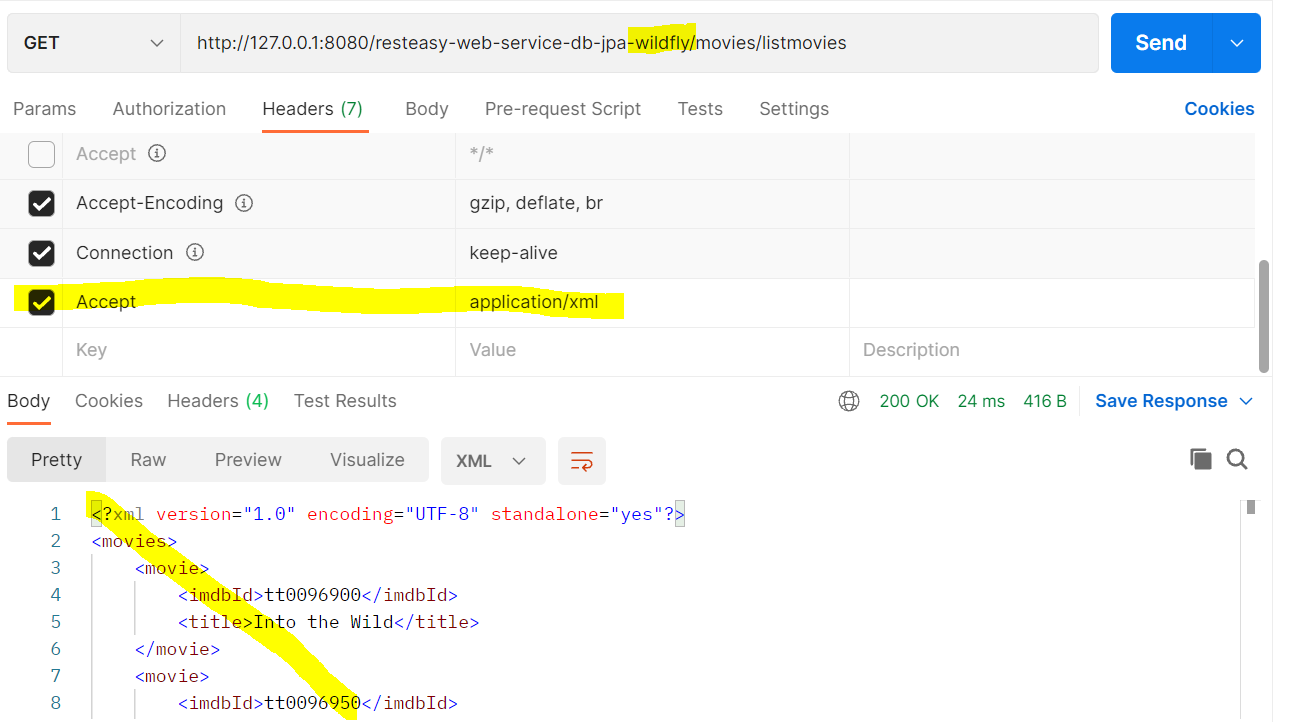
HTTP GET

http://127.0.0.1:8080/resteasy-web-service-db-wildfly/movies/listmovies

http\_request\_headers{

Accept: application/xml

}



HTTP GET

http://127.0.0.1:8080/resteasy-web-service-db-wildfly/movies/listmovies

http\_request\_headers{

Accept: application/json

}

