



CS230 - DISTRIBUIRANI SISTEMI

Java Persistence API

Lekcija 07

PRIRUČNIK ZA STUDENTE

# CS230 - DISTRIBUIRANI SISTEMI

## Lekcija 07

### *JAVA PERSISTENCE API*

- ✓ Java Persistence API
- ✓ Poglavlje 1: Kreiranje JPA entiteta
- ✓ Poglavlje 2: DAO (Data Access Objects) klase
- ✓ Poglavlje 3: Direktni inženjering - JPA entiteti iz tabela
- ✓ Poglavlje 4: Obeleženi upiti i JPQL
- ✓ Poglavlje 5: Validacija zrna
- ✓ Poglavlje 6: Relacije entiteta
- ✓ Poglavlje 7: Inverzni inženjering - baza podataka iz entiteta
- ✓ Poglavlje 8: Kreiranje JSF aplikacija iz JPA entiteta
- ✓ Poglavlje 9: Pokazni primer - JPA
- ✓ Poglavlje 10: Individualna vežba 7
- ✓ Poglavlje 11: Domaći zadatak 7
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

## ▼ Uvod

### UVOD

*Lekcija će se fokusirati na rad sa bazama podataka primenom Java Persistence API (JPA).*

Java Persistence API (JPA) predstavlja API za objektno - relaciono mapiranje (**object - relational mapping** - ORM). ORM alati pomažu prilikom automatizovanja mapiranja Java objekata u tabele relacionih baza podataka. Prve verzije J2EE koristile su entitetska zrna kao standardni ORM pristup. Entitetska zrna (**entity beans**) su uvek pokušavala da drže sinhronizovanim podatke koji se čuvaju u memoriji sa podacima baza podataka. Iako je ovo odlična ideja, u praksi je pokazala ozbiljne nedostatke koji su se reflektovali kroz loše performanse aplikacija.

U međuvremenu, razvijeno je nekoliko ORM API - ja, sa ciljem prevazilaženja ograničenja koja se dovode u vezu sa primenom entitetskih zrna. Između ostalih, najpoznatiji su: *Hibernate*, *EclipseLink*, iBatis, Cayenne i TopLink.

Sa Java EE 5 dolazi do deprekacije entitetskih zrna u korist JPA. JPA je integrisala ideje nekoliko ORM alata i postavila ih kao standard. Upravo tim alatima će se baviti ova lekcija.

Lekcija će se fokusirati na sledeće teme:

- Kreiranje JPA entiteta;
- Interakcija sa JPA entitetima primenom klase **EntityManager**;
- Generisanje JPA entiteta iz postojećih šema baza podataka;
- Korišćene JPA upita i **Java Persistence Query Language (JPQL)**;
- Kreiranje JSF aplikacije sa JPA entitetima.

Savladavanjem ove lekcije student će razumeti ORM pristup i ovladati primenom ORM alata u radu sa bazama podataka.

## ▼ Poglavlje 1

# Kreiranje JPA entiteta

## JPA ENTITET

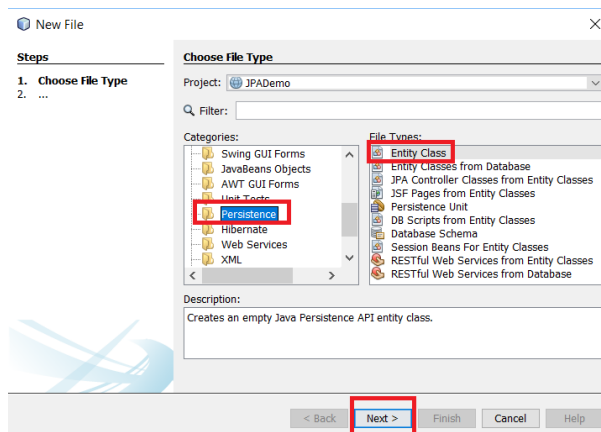
*Lekcija započinje izlaganje sa demonstracijom kreiranja prvog JPA entiteta.*

JPA entiteti su Java klase čija se polja čuvaju (perzistiraju) u bazama podataka pomoću JPA API. Navedene Java klase predstavljaju POJO (**Plain Old Java Objects**) objekte i kao takvi ne moraju da nasleđuju bilo kakvu roditeljski klasu ili da implementiraju bilo kakav specifični interfejs. **Java klasa koja se odnosi na JPA entitet je obeležena anotacijom @Entity.**

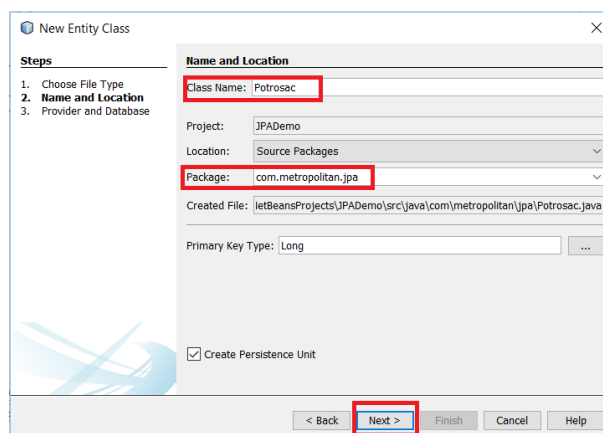
Lekcija će nastaviti praksu, uvedenu u prethodnim izlaganjima, da svaku analizu bazira na adekvatnom primeru. Takav slučaj će biti i ovde. Neophodno je kreirati novi veb projekat, primenom JavaServer Faces okvira, u okviru kojeg će biti implementiran prvi JPA entitet. Primenom razvojnog okruženja, neka to za potrebe demonstracije bude NetBeans IDE, biće kreiran projekat veb aplikacije pod nazivom JPADemo. Takođe, biće korišćen aplikativni server GlassFish.

U kreiranom projektu, sada je moguće kreirati prvu JPA klasu. U meniju File, bira se opcija New File. Otvara se dobro poznat prozor (videti sliku broj 1) u kojem se bira kategorija datoteke koja se kreira i odgovarajući tip datoteke. **Kao kategorija će biti izabrano Persistence, a tip datoteke će biti Entity Class.** Klikom na dugme Next otvara se prozor (videti sliku broj 2) u kojem se definiše naziv klase koja se kreira i paket kojem će klasa pripadati.

Klikom na Dugme Next, novog prozora, prelazi se na novi prozor **Provider and Database** o kojem će više diskusije biti u narednom izlaganju.



Slika 1.1 Izbor kategorije i tipa datoteke [izvor: autor]



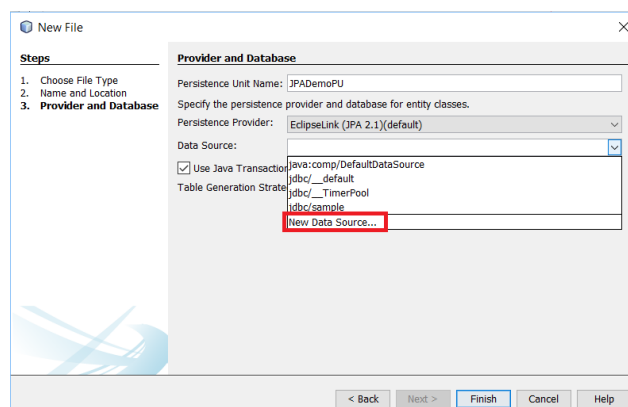
Slika 1.2 Definisanje naziva i paketa JPA klase [izvor: autor]

## PODEŠAVANJE PROVAJDERA

*EclipseLink je podrazumevana JPA implementacija za GlassFish aplikativni server.*

Projekat koji koristi JPA zahteva jedinicu perzistencije (persistence unit). Ona je definisana u fajlu pod nazivom persistence.xml. Razvojno okruženje NetBeans je automatski detektovao da ovaj fajl ne postoji pa je na prethodnoj slici čekirao opciju Create Persistence UNIT.

Klikom na dugme **Next**, prikazanog prozora u prethodnom izlaganju, prelazi se na novi prozor **Provider and Database** koji je prikazan sledećom slikom.



Slika 1.3 Provider and Database podešavanje [izvor: autor]

**Provider and Database** čarobnjak, prikazan slikom, predložiće naziv za jedinicu perzistencije koji u velikoj većini slučajeva može da bude prihvaćen.

NetBeans IDE podržava nekoliko JPA implementacija, kao što su: **EclipseLink**, **TopLink Essentials**, **Hibernate**, **KODO iOpenJPA**. **EclipseLink** je podrazumevana JPA implementacija za

GlassFish aplikativni server, pa će zbog toga, u ovom slučaju, biti izabrana kao provajder perzistencije (**Persistence Provider**), a to je pokazano prethodnom slikom.

Još jedan bitan zadatak je neophodno obaviti pre bilo kakve interakcije sa bazom podataka. Neophodno je podesiti tip baze podataka i izvora podataka na aplikativnom serveru.

## TIP BAZE PODATAKA

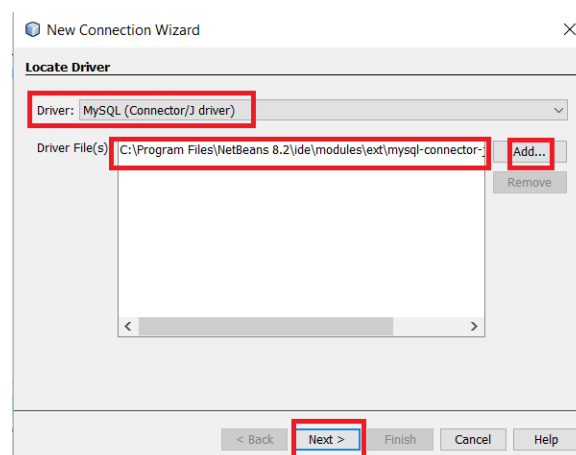
*Konekcije u connection pool-u nikad nisu zatvorene.*

Veoma bitan zadatak je obezbeđivanje informacija kojima je omogućeno konektovanje na bazu podataka, poput: naziva servera, porta, kredencijala i slično. Ove informacije su poznate pod nazivom **connection pool**. Prednost korišćenja ovakvog pristupa u radu sa bazama podataka, u odnosu na direktnu primenu JDBC, jeste u tome što konekcije u connection pool - u nikad nisu zatvorene i jednostavno se koriste u aplikaciji kada za njima postoji potreba. Ovim se značajno povećavaju performanse aplikacije budući da se izbegavaju zahtevne akcije, po pitanju performansi, otvaranja i zatvaranja konekcija.

Izvor podataka dozvoljava dobijanje informacija iz **connection pool** - a i poziv vlastite metode **getConnection()** za dobijanje konekcije sa bazom podataka iz **connection pool** - a. Ovde nije potrebno direktno obezbeđivanje reference na izvor podataka, **JPA** će to učiniti automatski.

Sada je neophodno vratiti se na prethodnu sliku. Za kreiranje novog izvora podataka neophodno je kliknuti na opciju **New Data Source** koja je istaknuta crvenom bojom u **Data Source ComboBox** kontroli.

Nakon izvršene navedene akcije, otvara se prozor pod nazivom **Locate Driver** (videti sledeću sliku) gde se bira tip drajvera za bazu podataka i odgovarajući JAR fajl (ukoliko nije ponuđen klikom na Add neophodno ga je pronaći i dodati).



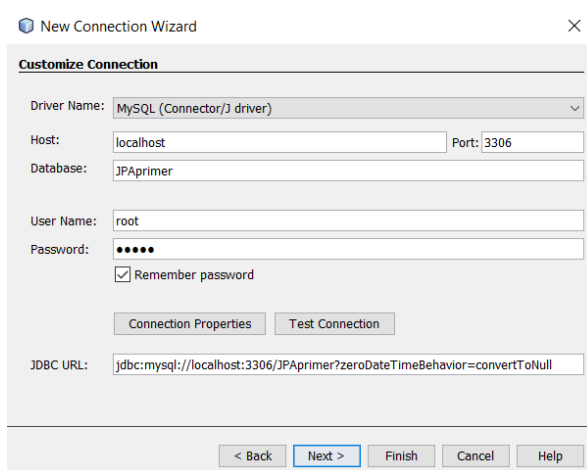
Slika 1.4 Izbor tipa i fajla drajvera baze podatak [izvor: autor]

Sa slike je moguće primetiti da je **izborpao** na MySQL tip baze podataka i njemu odgovarajući **drajver**. Klikom na dugme Next otvara se novi prozor, pod nazivom **Customize Connection** čija analiza sledi u narednom izlaganju.

## DODATNA PODEŠAVANJA

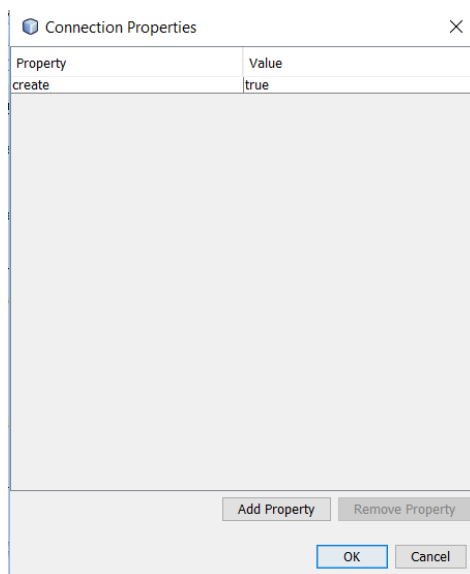
*Sledi izbor Host - a, Port - a i baze podataka iz koje će podaci biti korišćeni.*

Kao što je istaknuto u prethodnom izlaganju, dalja analiza započinje kod prozora **Customize Connections** koji je prezentovan sledećom slikom.



Slika 1.5 Customize Connections prozor [izvor: autor]

Ovde je još jednom istaknut tip drajvera baze podataka, nakon čega sledi izbor Host - a, Port - a i baze podataka iz koje će podaci biti korišćeni. Ovi podaci su bili automatski kreirani, osim naziva baze podataka kojeg korisnik sam određuje (JPAPrimer). Nakon toga unose se podaci, User Name i Password, neophodni za pristup bazi podataka (ukoliko postoje). Ukoliko baza podataka sa navedenim imenom, u ovom trenutku ne postoji, moguće je izvršiti njeno kreiranje klikom na opciju **Connection Properties** i podešavanjem opcije **create** na vrednost **true** (sledeća slika).



Slika 1.6 Connection Properties prozor [izvor: autor]

Nakon obavljanja ove akcije, moguće je još pogledati i polje za unos teksta koje ukazuje na JDBC URL. Nakon unosa imena baze podataka, odgovarajući String je automatski upisan u ovo polje.

Na kraju, biće neophodno uneti i JNDI ime, na primer jdbc/jpademo, čime će ova podešavanja biti završena.

## PRVI JPA ENTITET

*Nakon izvršenih podešavanja moguće je nastaviti kreiranja prvog JPA entiteta.*

Nakon izvršenih podešavanja moguće je nastaviti kreiranja prvog JPA entiteta na način koji je već prikazan slikama 1 i 2. Klasa poseduje naziv `Potrosac.java` i pripada paketu kome je dodeljen naziv `com.metropolitan.jpa` (pogledati sliku 2). Sadržaj datoteke je priložen sledećim inicijalnim listingom:

```
package com.ensode.jpaweb;
import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
public class Potrosac implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    public Long getId() {
        return id;
    }
    public void setId(Long id) {
        this.id = id;
    }
    //ostale metode (equals(), hashCode(), toString())
    izostavljene su zbog preglednosi
}
```

Ono što je prvo moguće uočiti, budući da se radi o entitetskoj klasi, jeste da je ona automatski obeležena anotacijom `@Entity`. To praktično znači da njoj direktno odgovara tabela sa identičnim nazivom iz kreirane baze podataka `JPAPrimer`.

Sledećom anotacijom `@id` je označeno koja polja u JPA entitetu čine primarni ključ tj, jednoznačni identifikator entiteta. Ne postoje dva entiteta sa identičnim primarnim ključevima. Postojanje navedenih anotacija je minimum koji klasa mora da ispunjava da bi mogla da bude smatrana entitetskom klasom. JPA dozvoljava automatsko generisanje primarnih ključeva. U tu svrhu je moguće koristiti anotaciju `@GeneratedValue`. Ovom anotacijom se definiše strategija kojom se generiše primarni ključ. U velikom broju slučajeva vrednost ove anotacije `GenerationType.AUTO` funkcioniše kako treba pa se uglavnom i



koristi (kao i u slučaju tekućeg primera). Moguće vrednosti za anotaciju **@GeneratedValue** date su sledećom tabelom.

Strategija generisanja primarnog ključa	Opis
<code>GenerationType.AUTO</code>	Provajder perzistencije će automatski izabrati strategiju generisanja primarnog ključa. Ovo je podrazumevana vrednost, ukoliko nije navedena.
<code>GenerationType.IDENTITY</code>	"Identity" kolona u tabeli baze podataka koja je mapirana sa JPA entitetom, mora biti upotrebljena za generisanje vrednosti primarnog ključa.
<code>GenerationType.SEQUENCE</code>	Sekvenca baze podataka se koristi za generisanje vrednosti primarnog ključa.
<code>GenerationType.TABLE</code>	Tabela baze podataka se koristi za generisanje vrednosti primarnog ključa.

Slika 1.7 Moguće vrednosti za anotaciju **@GeneratedValue** [izvor: autor]

## DODAVANJE PERZISTENTNIH POLJA U ENTITET

*Automatski generisan kod klase entiteta se proširuje odgovarajućim poljima i metodama.*

U nastavku, neophodno je u JPA entitet, odnosno odgovarajuću klasu, dodati polja koja će služiti da se u odgovarajućim kolonama, tabele koja odgovara JPA entitetu, čuvaju konkretne vrednosti u bazi podataka. Budući da su polja kreirana primenom modifikatora **private**, neophodno je u kod dodati i odgovarajuće **setter** i **getter** javne metode za manipulisanje ovim poljima.

Za konkretan primer, odmah ispod polja **id**, koje je automatski generisano, biće dodatak dva **String** polja **firstName** i **lastName** za ime i prezime kreiranih potrošača, respektivno. Za ova dva polja dodaju se i odgovarajuće **setter** i **getter** javne metode.

Kao rezultat javlja se klasa `Potrosac.java` čiji je modifikovan kod priložen u celosti sledećim listingom:

```
package com.metropolitan.jpa;

import java.io.Serializable;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

/**
 *
 * @author Vladimir Milicevic
 */
@Entity
public class Potrosac implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
```

```
@GeneratedValue(strategy = GenerationType.AUTO)
private Long id;
private String firstName;
private String lastName;

public String getFirstName() {
    return firstName;
}

public void setFirstName(String firstName) {
    this.firstName = firstName;
}

public String getLastName() {
    return lastName;
}

public void setLastName(String lastName) {
    this.lastName = lastName;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not
set
    if (!(object instanceof Potrosac)) {
        return false;
    }
    Potrosac other = (Potrosac) object;
    if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
```

```
public String toString() {  
    return "com.metropolitan.jpa.Potrosac[ id=" + id + " ]";  
}  
  
}
```

## ▼ Poglavlje 2

# DAO (Data Access Objects) klase

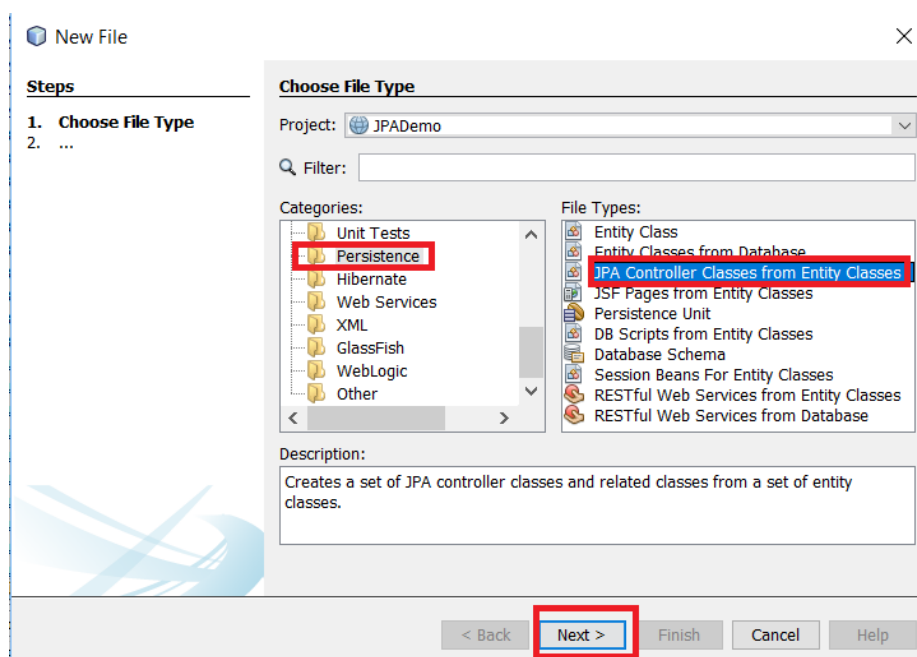
## KREIRANJE DAO KLASE

*Dobra praksa je korišćenje šablona dizajna za pisanje koda koji se obraća bazi podataka.*

Prilikom razvoja savremenih aplikacija koje podrazumevaju rad sa bazama podataka, veoma dobru praksu predstavlja korišćenje Data Access Objects (DAO) šablona dizajna za pisanje koda koji se obraća bazi podataka. DAO šabloni dizajna čuvaju sve funkcionalnosti pristupa bazi podataka unutar DAO klase. Na ovaj način se postiže jasno razdvajanje modula programa pomoću kojeg ostali slojevi aplikacije, poput logike korisničkog interfejsa ili poslovne logike, ne zavise od logike perzistencije.

Razvojno okruženje NetBeans IDE poseduje sposobnost direktnog kreiranja JPA klase kontrolera iz postojećih entiteta. Upravo u narednom izlaganju sledi kreiranje jedne ovakve klase i proširivanje započetog primera. Po dobro poznatom scenariju, koristeći razvojno okruženje, bira se opcija **File | New** i otvara se poznati prozor za izbor kategorije i tipa aplikacije. U ovom slučaju kao kategorija se bira **Persistence** kategorija, dok će **JPA Controller Classes from Entity Classes** predstavljati tip datoteke koja se kreira.

Sledećom slikom je prikazan početak, objašnjen u prethodnom izlaganju, kreiranja DAO klase.

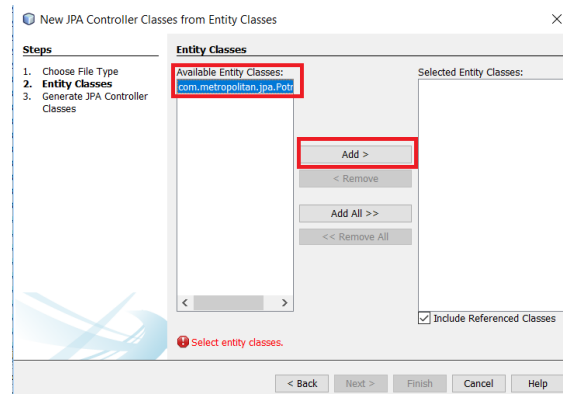


Slika 2.1 New File prozor za kreiranje DAO klase [izvor: autor]

## IZBOR ENTITETA

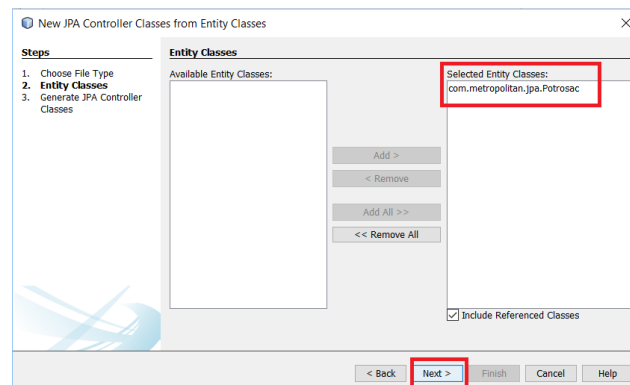
*Moguće je kreiranje DAO klase direktno iz entiteta.*

Kao što je istaknuto u prethodnom izlaganju, razvojno okruženje NetBeans IDE poseduje sposobnost direktnog kreiranja kontrolera JPA klase iz postojećih entiteta. Kada u prozoru, sa slike 1, usledi akcija korisnika na dugme **Next**, otvara se čarobnjak za izbor entiteta koji će poslužiti kao osnov za kreiranje JPA klase kontrolera. Navedeno je prikazano sledećom slikom.



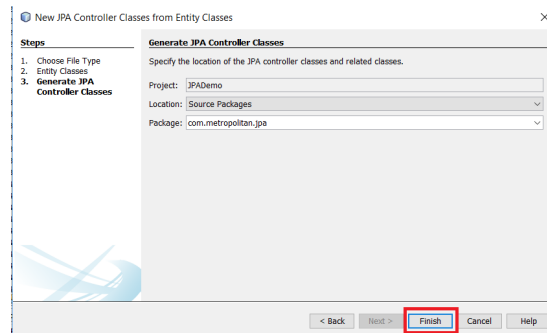
Slika 2.2 Izbor klase entiteta [izvor: autor]

Označavanjem klase u levom delu čarobnjaka, i klikom na dugme Add, u sredini, vrši se izbor entitetske klase pomoću koje će biti formirana kontroler klasa. Novo stanje u čarobnjaku je prikazano sledećom slikom.



Slika 2.3 Izabrana klasa entiteta [izvor: autor]

U nastavku, klikom na dugme **Next**, vrši se potvrđivanje ovog izbora i otvara prozor kojim se postupak kreiranja JPA klase kontrolera završava (klikom na **Finish**). Navedeno je prikazano sledećom slikom.



Slika 2.4 Poslednji korak u kreiranju JPA klase kontrolera [izvor: autor]

## LISTING SA OBJAŠNJENJEM

*Sledi prikaz i objašnjenje kreiranog kontrolera.*

Kao što može da se nasluti, iz prethodnog izlaganja, razvojno okruženje je na osnovu instrukcija dobijenih kroz navigaciju u prethodno prikazanom čarobnjaku, generisalo kod kontroler klase `PotrosacJpaController.java`, na osnovu entitetske klase `Potrosac.java`. Sledi listing koda klase:

```
package com.metropolitan.jpa;

import com.metropolitan.jpa.exceptions.NonexistentEntityException;
import com.metropolitan.jpa.exceptions.RollbackFailureException;
import java.io.Serializable;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Query;
import javax.persistence.EntityNotFoundException;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;
import javax.transaction.UserTransaction;

/**
 *
 * @author Vladimir Milicevic
 */
public class PotrosacJpaController implements Serializable {

    public PotrosacJpaController(UserTransaction utx,
        EntityManagerFactory emf) {
        this.utx = utx;
        this.emf = emf;
    }
    private UserTransaction utx = null;
    private EntityManagerFactory emf = null;

    public EntityManager getEntityManager() {
        return emf.createEntityManager();
    }
}
```

```

    }

    public void create(Potrosac customer) throws
        RollbackFailureException, Exception {
        EntityManager em = null;
        try {
            utx.begin();
            em = getEntityManager();
            em.persist(customer);
            utx.commit();
        } catch (Exception ex) {
            try {
                utx.rollback();
            } catch (Exception re) {
                throw new RollbackFailureException(
                    "An error occurred attempting to roll back the transaction.", re );
            }
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

    public void edit(Potrosac customer) throws
        NonexistentEntityException, RollbackFailureException, Exception {
        EntityManager em = null;
        try {
            utx.begin();
            em = getEntityManager();
            customer = em.merge(customer);
            utx.commit();
        } catch (Exception ex) {
            try {
                utx.rollback();
            } catch (Exception re) {
                throw new RollbackFailureException(
                    "An error occurred attempting to roll back the transaction.", re);
            }
        }
        String msg = ex.getLocalizedMessage();
        if (msg == null || msg.length() == 0) {
            Long id = customer.getId();
            if (findCustomer(id) == null) {
                throw new NonexistentEntityException(
                    "The customer with id " + id
                    + " no longer exists.");
            }
        }
        throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

```

```

        }
    }
}

public void destroy(Long id) throws NonexistentEntityException,
    RollbackFailureException, Exception {
    EntityManager em = null;
    try {
        utx.begin();
        em = getEntityManager();
        Potrosac customer;
        try {
            customer = em.getReference(Potrosac.class, id);
            customer.getId();
        } catch (EntityNotFoundException enfe) {
            throw new NonexistentEntityException(
                "The customer with id " + id
                + " no longer exists.", enfe);
        }
        em.remove(customer);
        utx.commit();
    } catch (Exception ex) {
        try {
            utx.rollback();
        } catch (Exception re) {
            throw new RollbackFailureException(
                "An error occurred attempting to roll back the transaction.", re);
        }
    }
    throw ex;
    } finally {
        if (em != null) {
            em.close();
        }
    }
}

public List<Potrosac> findCustomerEntities() {
    return findCustomerEntities(true, -1, -1);
}

public List<Potrosac> findCustomerEntities(int maxResults,
    int firstResult) {
    return findCustomerEntities(false, maxResults, firstResult);
}

private List<Potrosac> findCustomerEntities(boolean all, int maxResults,
    int firstResult) {
    EntityManager em = getEntityManager();
    try {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        cq.select(cq.from(Potrosac.class));
        Query q = em.createQuery(cq);
        if (!all) {

```



```

        q.setMaxResults(maxResults);
        q.setFirstResult(firstResult);
    }
    return q.getResultList();
} finally {
    em.close();
}
}

public Potrosac findCustomer(Long id) {
    EntityManager em = getEntityManager();
    try {
        return em.find(Potrosac.class, id);
    } finally {
        em.close();
    }
}

public int getCustomerCount() {
    EntityManager em = getEntityManager();
    try {
        CriteriaQuery cq = em.getCriteriaBuilder().createQuery();
        Root<Potrosac> rt = cq.from(Potrosac.class);
        cq.select(em.getCriteriaBuilder().count(rt));
        Query q = em.createQuery(cq);
        return ((Long) q.getSingleResult()).intValue();
    } finally {
        em.close();
    }
}
}

```

Kao što je moguće primetiti kreirane su CRUD metode za JPA entitete. Metoda koja kreira nove entitete naziva se **create()** i uzima, kao jedini argument, instancu JPA entiteta. Metoda jednostavno poziva metodu **persist()** za **EntityManager** objekat čime se brine o čuvanju podataka entiteta u bazi podataka.

CRUD operacija **read** realizovana je pomoću nekoliko operacija. Metoda **findCustomer()** preuzima primarni ključ JPA entiteta, koji bi trebalo da bude vraćen, kao jedini argument i poziva **find()** metodu objekta **EntityManager** za preuzimanje podataka iz baze podataka i vraćanje instance JPA klase. Generisano je i nekoliko preklopljenih **findCustomerEntities()** metoda pomoću kojih je moguće vratiti više od jednog entiteta iz baze podataka. Jedna od njih ima sledeći oblik:

```

private List<Customer> findCustomerEntities(boolean all, int maxResults, int
firstResult)

```

Prvi parametar je **Boolean** tipa i ukazuje da li se žele preuzeti sve vrednosti iz baze podataka. Drugi parametar određuje najveći broj rezultata koje je moguće preuzeti, a poslednji parametar ukazuje na rezultat koji će prvi biti preuzet.

Metodom `edit()` se vrši ažuriranje postojećih entiteta, pozivom metode `merge()`, objekta `EntityManager`, koja preuzima kao parametar JPA entitet kojim ažurira bazu podataka.

Konačno, entitet se briše metodom `destroy()`.

## ▼ Poglavlje 3

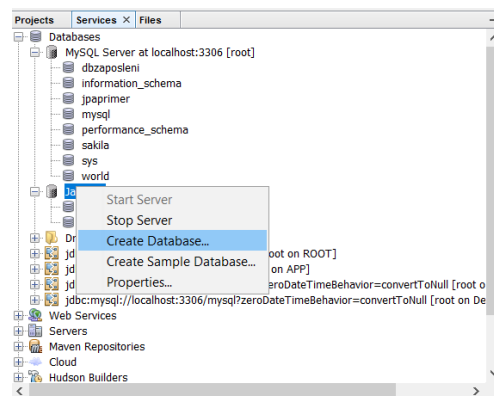
# Direktni inženjering - JPA entiteti iz tabela

## KREIRANJE I / ILI PRISTUP BAZI PODATAKA

*Savremeni alati dozvoljavaju generisanje JPA entiteta iz već postojećih šema baza podataka.*

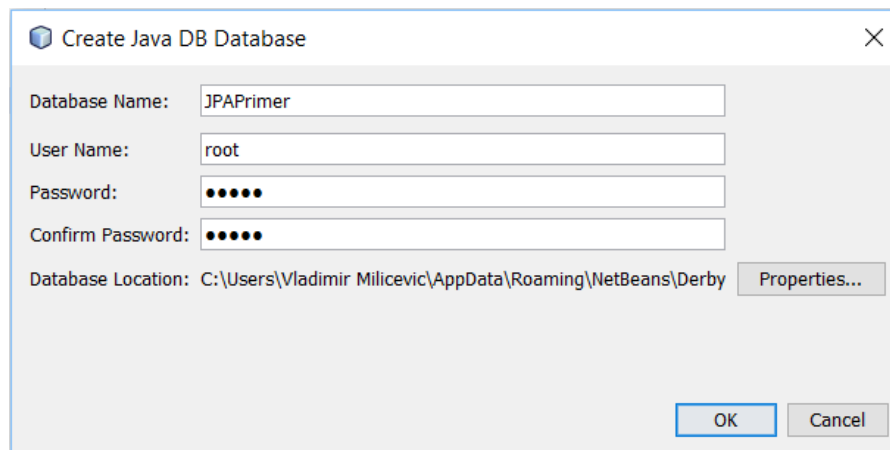
Kada se rade aplikacije nad bazama podataka, u velikom broju slučajeva, koriste se baze podataka koje su već kreirane od strane administratora baze podataka. Savremeni alati dozvoljavaju generisanje JPA entiteta iz već postojećih šema baza podataka i nataj način olakšavaju rad i smanjuju napor programera prilikom rada na kreiranju datoteka neke veb aplikacije.

U ovom delu lekcije, oslanjajući se na konkretne primere, biće analiziran i demonstriran način kreiranja JPA entiteta iz postojeće baze podataka, koja je uvedena i korišćena u primeru iz prethodnog izlaganja. Da bi ovo bilo obavljeno neophodno je otvoriti, primenom razvojnog okruženja, postojeći projekat i u tabu Services izvršiti proširenje kreirane baze podataka. Ovde se desnim klikom bira opcija JAVA DB, a zatim i Create DB (sledeća slika).



Slika 3.1 Kreiranje nove baze podataka [izvor: autor]

Sada je neophodno uneti informacije o bazi podataka u otvorenom čarobnjaku Create Java DB Database, kao na sledećoj slici.



Slika 3.2 Podaci o bazi podataka [izvor: autor]

Sada je sve spremno za izvršavanje SQL skriptova kojima će biti kreirane tabele baze podataka.

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## KREIRANJE NOVIH TABELA U POSTOJEĆOJ BAZI PODATAKA

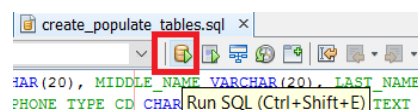
*Izvršavanje SQL upita biće kreirane tabele u bazi podataka.*

Nad postojećom bazom podataka JPAPrimer, moguće je sada izvršiti brojne SQL skriptove od kojih će većina, u početku, da se odnosi na kreiranje tabela baze podataka. Da bi posao bio olakšan dodatno, preuzet je fajl pod nazivom `create_populate_tables.sql` kao dodatni materijal uz preporučeni udžbenik (videti literaturu pod 2) koji je snabdeven SQL upitima za kreiranje i popunjavanje baze podataka. Sledi deo koda, koji se odnosi na kreiranje tabela:

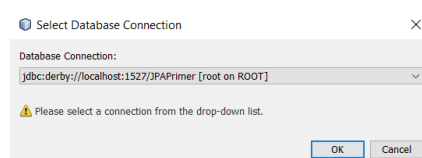
```
CREATE TABLE CUSTOMER (CUSTOMER_ID INT PRIMARY KEY, FIRST_NAME VARCHAR(20),
MIDDLE_NAME VARCHAR(20), LAST_NAME VARCHAR(20), EMAIL VARCHAR(30));
CREATE TABLE TELEPHONE_TYPE(TELEPHONE_TYPE_ID INT PRIMARY KEY, TELEPHONE_TYPE_CD
CHAR(1), TELEPHONE_TYPE_TEXT VARCHAR(20));
CREATE TABLE TELEPHONE (TELEPHONE_ID INT PRIMARY KEY, TELEPHONE_TYPE_ID INT
REFERENCES TELEPHONE_TYPE, CUSTOMER_ID INT REFERENCES CUSTOMER, TELEPHONE_NUMBER
CHAR(12));
CREATE TABLE ADDRESS_TYPE (ADDRESS_TYPE_ID INT PRIMARY KEY, ADDRESS_TYPE_CODE
CHAR(1), ADDRESS_TYPE_TEXT VARCHAR(20));
CREATE TABLE US_STATE(US_STATE_ID INT PRIMARY KEY, US_STATE_CD CHAR(2) NOT NULL,
US_STATE_NM VARCHAR(30) NOT NULL);
CREATE TABLE "APP"."US_CITY"(US_CITY_ID int PRIMARY KEY NOT NULL,US_CITY_NM
varchar(30),ZIP char(5),US_STATE_ID int);
ALTER TABLE US_CITY ADD CONSTRAINT SQL080413034555231 FOREIGN KEY (US_STATE_ID)
REFERENCES US_STATE(US_STATE_ID) ON DELETE NO ACTION ON UPDATE NO ACTION;
CREATE INDEX SQL080413034555231 ON US_CITY(US_STATE_ID);
CREATE UNIQUE INDEX SQL080413034555230 ON US_CITY(US_CITY_ID);
```

```
CREATE TABLE ADDRESS (ADDRESS_ID INT PRIMARY KEY, ADDRESS_TYPE_ID INT REFERENCES
ADDRESS_TYPE, CUSTOMER_ID INT REFERENCES CUSTOMER,
ADDR_LINE_1 VARCHAR(100), ADDR_LINE_2 VARCHAR(100), CITY VARCHAR (100), US_STATE_ID
INT REFERENCES US_STATE, ZIP CHAR(5));
CREATE TABLE CUSTOMER_ORDER(CUSTOMER_ORDER_ID INT PRIMARY KEY, CUSTOMER_ID INT
REFERENCES CUSTOMER, ORDER_NUMBER VARCHAR (10), ORDER_DESCRIPTION VARCHAR(200));
CREATE TABLE ITEM(ITEM_ID INT PRIMARY KEY, ITEM_NUMBER VARCHAR(10), ITEM_SHORT_DESC
VARCHAR(100), ITEM_LONG_DESC VARCHAR(500));
CREATE TABLE ORDER_ITEM(CUSTOMER_ORDER_ID INT REFERENCES CUSTOMER_ORDER, ITEM_ID
INT REFERENCES ITEM);
CREATE TABLE APP_USER(APP_USER_ID INT PRIMARY KEY, USER_NAME VARCHAR(10), PASSWORD
VARCHAR (15));
CREATE TABLE USER_ROLE(ROLE_ID INT PRIMARY KEY, ROLE_NAME VARCHAR(10));
CREATE TABLE APP_USER_ROLE(APP_USER_ID INT REFERENCES APP_USER, USER_ROLE_ID INT
REFERENCES USER_ROLE);
CREATE TABLE "APP"."SEQUENCE"
(
    SEQ_NAME varchar(50),
    SEQ_COUNT decimal(15)
)
```

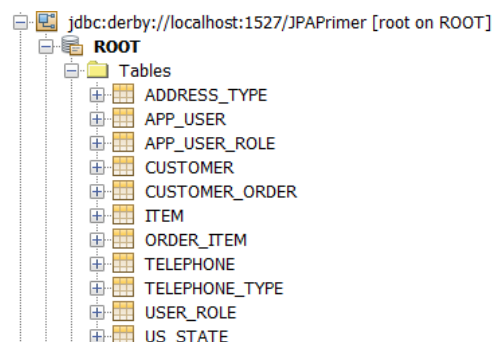
U projektu se jednostavno dolazi do ovog fajla. Bira se **File | Open File**, a zatim se datoteka pronalazi na disku. Kada je ova datoteka, sa SQL upitima, učitana u projekat, primenom razvojnog okruženja je neophodno je izvršiti je klikom na ikonicu **Run SQL** kao što je prikazano na Slici 3. Otvara se prozor za izbor baze podataka nad kojom će upit biti izvršen (Slika 4).



Slika 3.3 Izvršavanje SQL skript [izvor: autor]



Slika 3.4 Izbor baze podataka za izvršenje SQL skripta [izvor: autor]



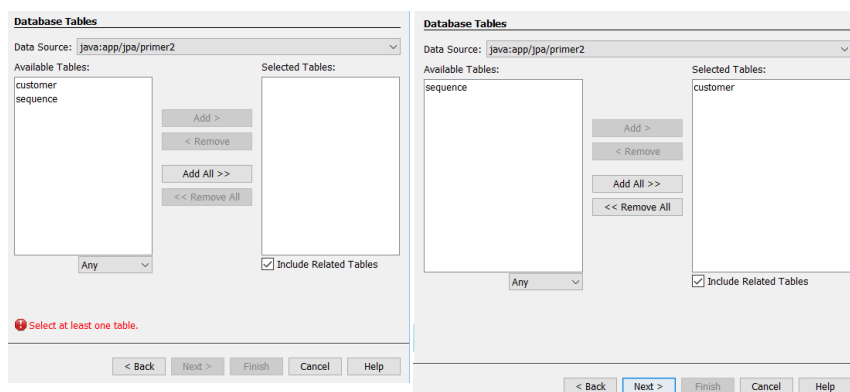
Slika 3.5 Tabele su kreirane i popunjene [izvor: autor]

## KREIRANJE JPA ENTITETA IZ POSTOJEĆE ŠEME BAZE PODATAKA

*Iz tabela baze potaka je moguće automatski generisati JPA entitete.*

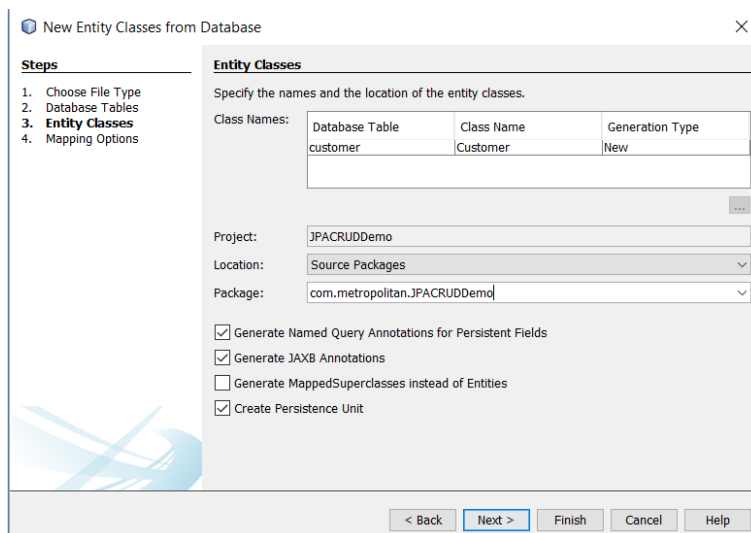
Za početak će biti, na dobro poznat način kreiran veb projekat pod nazivom, **JPACRUDDemo**. U novom projektu će biti izabrana opcija **File | New**, a zatim se kreira datoteka iz kategorije **Persistence** i tipa **Entity Classes from Database** (o svemu ovome je već bilo govora).

U prozoru **New Entity Classes from Database** biraju se tabele od kojih je neophodno kreirati JPA entitete (sledeća slika).

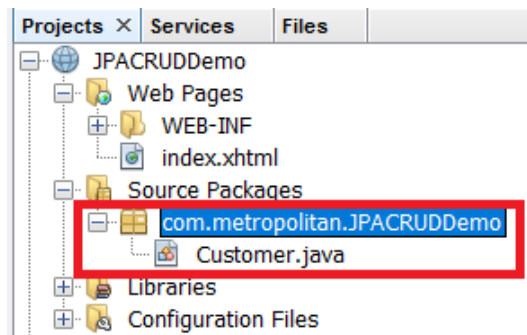


Slika 3.6 Izbor tabele baze podataka za kreiranje JPA entiteta [izvor: autor]

Za potrebe primera, biće izabrana samo jedna tabela **customer** markiranjem i klikom na dugme **Add** (vidi sliku gore). Klikom na dugme **Next**, ide se na naredni prozor u kojem se nalaze sve informacije u vezi sa klasom entiteta koja se kreira, uz napomenu da je neophodno uneti naziv paketa kojem će klasa da pripada (slika 7). Posao se završava klikom na dugme **Finish**.



Slika 3.7 Informacije o klasi entiteta koja se kreira [izvor: autor]



Slika 3.8 Klasa entiteta je kreirana [izvor: autor]

## DODATNA RAZMATRANJA

*Automastki generisan kod JPA klase zahteva dodatnu analizu.*

Na prethodnoj slici je moguće primetiti sa se u folderu **Source Packages** projekta pojavila Java klasa pod nazivom **Customer.java**. Datoteka odgovara kreiranom JPA entitetu iz tabele baze podataka **JPAPrimer** i njen kod je priložen sledećim listingom:

```
package com.metropolitan.JPACRUDDemo;

import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import javax.xml.bind.annotation.XmlRootElement;

/**
 *
 * @author Vladimir Milicevic
 */
@Entity
@Table(name = "customer")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Customer.findAll", query = "SELECT c FROM Customer c")
    , @NamedQuery(name = "Customer.findById", query = "SELECT c FROM Customer c WHERE c.id = :id")
    , @NamedQuery(name = "Customer.findByFirstname", query = "SELECT c FROM Customer c WHERE c.firstname = :firstname")
    , @NamedQuery(name = "Customer.findByLastname", query = "SELECT c FROM Customer c WHERE c.lastname = :lastname")})
public class Customer implements Serializable {
```

```
private static final long serialVersionUID = 1L;
@Id
@Basic(optional = false)
@NotNull
@Column(name = "ID")
private Long id;
@Size(max = 255)
@Column(name = "FIRSTNAME")
private String firstname;
@Size(max = 255)
@Column(name = "LASTNAME")
private String lastname;

public Customer() {
}

public Customer(Long id) {
    this.id = id;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}

public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
```



```
// TODO: Warning - this method won't work in the case the id fields are not
set
    if (!(object instanceof Customer)) {
        return false;
    }
    Customer other = (Customer) object;
    if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "com.metropolitan.JPACRUDDemo.Customer[ id=" + id + " ]";
}
}
```

Za početak je moguće primetiti da je klasa obeležena anotacijom **@Entity** što nedvosmisleno ukazuje da se radi o JPA entitetu. Dalje, anotacijom **@Table** *istaknuta* je tabela sa kojom je JPA entitet povezan.

Takođe, moguće je primetiti da je jedno polje automatski obeleženo anotacijom **@Id**. Ovo polje je odgovaralo primarnom ključu tabele iz koje je generisan JPA entitet. U ovom slučaju se ne primenjuje strategija generisanja primarnog ključa i anotaciju **@GeneratedValue** je neophodno manuelno implementirati. Anotacijom **@Basic** su polja označena kao neopcionalna.

Anotacijama **@Column** su polja JPA entiteta povezana sa odgovarajućim kolonama tabele baze podataka.

O upitima, koji su obeleženi anotacijom, **@NamedQueries** biće više diskusije u izlaganju koje sledi.

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ Poglavlje 4

# Obeleženi upiti i JPQL

## PRIMENA OBELEŽENIH UPITA

*Anotacijom `@NamedQueries` obeležena je lista upita koje izvršava kao obeleženi upit.*

Za početak neophodno je, za potrebe izlaganja, izolovati deo koda iz prethodno kreirane [JPA](#) klase. Kod je priložen sledećim listingom:

```
@NamedQueries({  
  
    @NamedQuery(name = "Customer.findAll", query = "SELECT c FROM Customer c")  
  
    , @NamedQuery(name = "Customer.findById", query = "SELECT c FROM Customer c  
WHERE c.id = :id")  
  
    , @NamedQuery(name = "Customer.findByFirstname", query = "SELECT c FROM  
Customer c WHERE c.firstname = :firstname")  
  
    , @NamedQuery(name = "Customer.findByLastname", query = "SELECT c FROM Customer  
c WHERE c.lastname = :lastname")  
})
```

Akcentat je na primeni anotacije `@NamedQueries` koja sadrži **value** atribut. Vrednost ovog atributa predstavlja niz `@NamedQuery` anotacija koje prihvataju dva argumenta **name** i **query**. Atributom **name** određen je logički naziv koji se pridružuje anotaciji (po konvenciji naziv JPA klase se koristi kao deo naziva upita), dok je atributom **query** priložen [JPQL](#) upit kojeg će izvršiti obeleženi upit.

Anotacija `@NamedQueries` će biti generisana samo u slučaju kada se klikne na CheckBox pod nazivom Generate Named Query Annotations for Persistent Fields u čarobnjaku New Entity Classes from Database.

JPQL je JPA specifični jezik upita čija je sintaksa veoma slična SQL jeziku. Čarobnjak New Entity Classes from Database generiše JPQL upit za svako polje u entitetu. Kada se upit izvrši biće vraćena lista svih instanci entiteta koje odgovaraju zadatom kriterijumu upita.

Sledećim kodom je priložena jedna DAO klasa:

```
import java.util.List;  
import javax.persistence.EntityManager;
```

```
import javax.persistence.Query;
public class CustomerDAO {
    public List findCustomerByLastName(String someLastName)
    {
        //deo koda je izostavljen zbog jasnoće i preglednosti

        Query query =
        em.createNamedQuery("Customer.findByLastName");
        query.setParameter("lastName", someLastName);
        List resultList = query.getResultList();
        return resultList;
    }
}
```

Moguće je primetiti da DAO objekat sadrži metodu koja će vratiti listu **Customer** entiteta čije prezime odgovara vrednosti parametra **lastName**. Da bi navedeno bilo realizovano, neophodno je obezbediti instancu klase **javax.persistence.Query**. Ovo je obavljeno angažovanjem metode **createNamedQuery()** **EntityManager** objekta i prosleđivanjem naziva upita.

Kada su određeni svi parametri upita, pozivom metode **getResultList()**, **Query** objekta, vraćaju se svi rezultati koji odgovaraju kriterijumu upita.

## ▼ Poglavlje 5

# Validacija zrna

## ANOTACIJE ZA PROVERU PODATAKA

*Validacija zrna je uvedena sa Java EE 6.*

Validacija zrna ([bean validation](#)) je uvedena sa Java EE 6 kao deo specifikacije Java [Specification Request \(JSR 303\)](#). Specifikacija provere zrna uključuje skup anotacija kojima je moguće vršiti proveru podataka. Koristeći čarobnjak za generisanje JPA klasa, razvojno okruženje NetBeans je u potpunosti iskoristilo prednosti koje validacije zrna nosi, dodavanjem navedenih anotacija odgovarajućim poljima koja odgovaraju kolonama tabele baze podataka iz koje je generisana JPA klasa.

U narodnom izlaganju biće prezentovane neke anotacije za proveru podataka koje su integrisanu u [Customer](#) entitet. Polje [customerId](#) je obeleženo anotacijom [@NotNull](#) koja ne dozvoljava polju da prihvati [null](#) vrednost.

Nekoliko polja entiteta [Customer](#) je obeleženo anotacijom [@Size](#) koja ukazuje na maksimalan broj karaktera kojih osobina zrna može da prihvati. Ova vrednost je dobijena iz tabele kojom je entitet generisan.

Biće analizirano još nekoliko validacionih anotacija. Na primer, anotacija [@Pattern](#) obezbeđuje da se obeleženo polje podudara sa datim regularnim iskazom.

Sledećim listingom je izolovan kod iz JPA klase entiteta sa linijama koda koje sadrže anotacije za proveru vrednosti polja.

```
@Id
@Basic(optional = false)
@NotNull
@Column(name = "CUSTOMER_ID")
private Integer customerId;
@Size(max = 20)
@Column(name = "FIRST_NAME")
private String firstName;
@Size(max = 20)
@Column(name = "MIDDLE_NAME")
private String middleName;
@Size(max = 20)
@Column(name = "LAST_NAME")
private String lastName;
// @Pattern(regexp="[a-z0-9!#$%&'*/+=?^_`{|}~-]+(?:\\.
[a-z0-9!#$%&'*/+=?^_`{|}~-]+)*(@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\\.)+[a-z0-9](?:[a-z0-9-]*[a-z0-9])?)?", message="Invalid email")//
```

```
if the field contains email address consider using this annotation to
enforce field validation
@Size(max = 30)
@Column(name = "EMAIL")
private String email;
```

## ▼ Poglavlje 6

# Relacije entiteta

## @ONETOMANY I @MANYTOONE ANOTACIJE

*Za definisanje relacija između JPA entiteta moguće je primeniti nekoliko anotacija.*

Za definisanje relacija između JPA entiteta moguće je primeniti nekoliko anotacija. Za potrebe analize i izlaganja, ponovo će u centru pažnje biti JPA entitet **Customer** kojem je demonstrirani čarobnjak za kreiranje JPA stranica mogao da dodeli izvesne anotacije koje ukazuju na kardinalnost relacija CUSTOMER tabele iz baze podataka.

Prva anotacija koja će biti obrađena je @OneToMany. Svako polje obeleženo anotacijom @OneToMany pripada tipu podataka java.util.Collection. Na jednoj strani relacije je entitet Customer (potrošač) koji može da ima više porudžbina, adresa, telefonskih bojeva i slično. U ovom scenariju rada sa kolekcijama, čarobnjak koristi generike za specificiranje tipova objekata koje je moguće dodati u kolekciju. Objekti ovih kolekcija su JPA entiteti povezani sa odgovarajućim tabelama iz aktuelne šeme baze podataka. Anotacija **@OneToMany** poseduje atribut **mappedBy**. Vrednost ovog atributa mora da odgovara nazivu polja sa druge strane relacije.

Drugu stranu relacije može da čini entitet **Address**. Ovaj entitet će svojim listingom da demonstrira mogućnost posedovanja više adresa za jednog potrošača. Entitet će biti prikazan odgovarajućim listingom u kojem se ističe još jedna anotacija **@JoinColumn**. Atribut **name** ove anotacije ukazuje na kolonu u bazi podataka sa kojom je entitet mapiran za definisanje ograničenja stranog ključa između tabela **ADDRESS** i **CUSTOMER** (konkretno, CUSTOMER\_ID kolona za ADDRESS tabelu). Sledi listing JPA klase **Address.java** koji sadrži opisane linije koda:

```
package com.metropolitan.jpa;
import java.io.Serializable;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
@Entity
@Table(name = "ADDRESS")
```

```

@NamedQueries({
@NamedQuery(name = "Address.findAll",
query = "SELECT a FROM Address a"),
@NamedQuery(name = "Address.findById",
query = "SELECT a FROM Address a WHERE a.addressId =
:addressId"),
@NamedQuery(name = "Address.findByAddrLine1",
query = "SELECT a FROM Address a WHERE a.addrLine1 =
:addrLine1"),
@NamedQuery(name = "Address.findByAddrLine2",
query = "SELECT a FROM Address a WHERE a.addrLine2 =
:addrLine2"),
@NamedQuery(name = "Address.findByCity",
query = "SELECT a FROM Address a WHERE a.city = :city"),
@NamedQuery(name = "Address.findByZip",
query = "SELECT a FROM Address a WHERE a.zip = :zip"))
public class Address implements Serializable {
private static final long serialVersionUID = 1L;
@Id
@Basic(optional = false)
@NotNull
@Column(name = "ADDRESS_ID")
private Integer addressId;
@Size(max = 100)
@Column(name = "ADDR_LINE_1")
private String addrLine1;
@Size(max = 100)
@Column(name = "ADDR_LINE_2")
private String addrLine2;
@Size(max = 100)
@Column(name = "CITY")
private String city;
@Size(max = 5)
@Column(name = "ZIP")
private String zip;
@JoinColumn(name = "ADDRESS_TYPE_ID",
referencedColumnName = "ADDRESS_TYPE_ID")
@ManyToOne
private AddressType addressType;
@JoinColumn(name = "CUSTOMER_ID",
referencedColumnName = "CUSTOMER_ID")
@ManyToOne
private Customer customer;
@JoinColumn(name = "US_STATE_ID",
referencedColumnName = "US_STATE_ID")
@ManyToOne
private UsState usState;
//metode i konstruktori zbog jasnoće i preglednosti su izostavljeni
}

```

## ANOTACIJE RELACIJA - MANY-TO-MANY RELACIJA

*Pored podrške za obrađene relacije, JPA ima podršku i za many-to-many i one-to-one relacije.*

Kao nastavak na izlaganje o anotacijama `@OneToMany` i `@ManyToOne` naslanja se diskusija na temu anotacije `@ManyToMany`. Na primer, baza podataka, koja je predmet analize, može da sadrži tabele `CUSTOMER_ORDER` i `ITEM` u kojima se čuvaju podaci o porudžbinama potrošača i stavkama porudžbine, respektivno. Jedna porudžbina može da sadrži više proizvoda i jedan proizvod može da bude sadržan u više porudžbina.

Sada se prilaže kodovi datoteka `Item.java` i `CustomerOrder.java` koje odgovaraju JPA entitetu koji je deo many-to-many relacije.

```
package commetropolitan.jpa;
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.ManyToMany;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
@Entity
@Table(name = "ITEM")
@NamedQueries({
    @NamedQuery(name = "Item.findAll", query = "SELECT i FROM Item i"),
    @NamedQuery(name = "Item.findById", query = "SELECT i FROM Item i WHERE i.itemId = :itemId"),
    @NamedQuery(name = "Item.findByIdItemNumber", query = "SELECT i FROM Item i WHERE i.itemNumber = :itemNumber"),
    @NamedQuery(name = "Item.findByIdItemShortDesc", query = "SELECT i FROM Item i WHERE i.itemShortDesc = :itemShortDesc"),
    @NamedQuery(name = "Item.findByIdItemLongDesc", query = "SELECT i FROM Item i WHERE i.itemLongDesc = :itemLongDesc")})
public class Item implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "ITEM_ID")
    private Integer itemId;
    @Size(max = 10)
    @Column(name = "ITEM_NUMBER")
    private String itemNumber;
    @Size(max = 100)
    @Column(name = "ITEM_SHORT_DESC")
```



```
private String itemShortDesc;
@Size(max = 500)
@Column(name = "ITEM_LONG_DESC")
private String itemLongDesc;
@ManyToMany(mappedBy = "itemCollection")
private Collection<CustomerOrder> customerOrderCollection;
//metode i konstruktori izostavljeni zbog preglednosti.
}
```

```
package com.metropolitan.jpa;
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.Basic;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.JoinTable;
import javax.persistence.ManyToMany;
import javax.persistence.ManyToOne;
import javax.persistence.NamedQueries;
import javax.persistence.NamedQuery;
import javax.persistence.Table;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
@Entity
@Table(name = "CUSTOMER_ORDER")
@NamedQueries({
    @NamedQuery(name = "CustomerOrder.findAll",
        query = "SELECT c FROM CustomerOrder c"),
    @NamedQuery(name = "CustomerOrder.findById",
        query = "SELECT c FROM CustomerOrder c WHERE c.customerOrderId = :customerOrderId"),
    @NamedQuery(name = "CustomerOrder.findbyOrderNumber",
        query = "SELECT c FROM CustomerOrder c WHERE c.orderNumber = :orderNumber"),
    @NamedQuery(name = "CustomerOrder.findbyOrderDescription",
        query = "SELECT c FROM CustomerOrder c WHERE c.orderDescription = :orderDescription"))
public class CustomerOrder implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Column(name = "CUSTOMER_ORDER_ID")
    private Integer customerOrderId;
    @Size(max = 10)
    @Column(name = "ORDER_NUMBER")
    private String orderNumber;
    @Size(max = 200)
    @Column(name = "ORDER_DESCRIPTION")
    private String orderDescription;
```

```
@JoinTable(name = "ORDER_ITEM", joinColumns = {
@JoinColumn(name = "CUSTOMER_ORDER_ID",
referencedColumnName = "CUSTOMER_ORDER_ID")},
inverseJoinColumns = {
@JoinColumn(name = "ITEM_ID",
referencedColumnName = "ITEM_ID")})
@ManyToMany
private Collection<Item> itemCollection;
@JoinColumn(name = "CUSTOMER_ID", referencedColumnName =
"CUSTOMER_ID")
@ManyToOne
private Customer customer;
//konstruktori i metode su izostavljeni iz razloga preglednosti
}
```

## ANOTACIJE RELACIJA - ONE-TO-ONE RELACIJA

*Neophodno je obraditi još jednu anotaciju.*

Na kraju moguće je kreirati one-to-one tip relacije između dva JPA entiteta. Strana koja poseduje relaciju mora da poseduje polje JPA entiteta sa druge strane relacije i ova polja moraju da budu obeležena anotacijama **@OneToOne** i **@JoinColumn**.

U nedostatku adekvatne ilustracije, u tekućem primeru, za relaciju tipa **@OneToOne** biće kreirane dve nove datoteke kojima se demonstrira primena ove anotacije.

Na primer, osoba oblači košulju. Svaka osoba poseduje jedno dugme na pupku, jedno dugme na pupku može biti na jednoj osobi. To je prikazano sledećim datotekama.

```
@Entity
public class Person implements Serializable {
@JoinColumn(name="BELLY_BUTTON_ID")
@OneToOne
private BellyButton bellyButton;
public BellyButton getBellyButton(){
return bellyButton;
}
public void setBellyButton(BellyButton bellyButton){
this.bellyButton = bellyButton;
}
}
```

```
@Entity
@Table(name="BELLY_BUTTON")
public class BellyButton implements Serializable{
{
@OneToOne(mappedBy="bellyButton")
private Person person;
public Person getPerson(){
return person;
}
```

```
}  
public void getPerson(Person person){  
    this.person=person;  
}  
}
```

## ▼ Poglavlje 7

# Inverzni inženjering - baza podataka iz entiteta

## OBRNUTO RAZMIŠLJANJE - KREIRANJE BAZE PODATAKA IZ ENTITETA

*Kod inverznog inženjeringa programer kreira entitetske klase iz kojih se kreiraju tabele baze podataka.*

U dosadašnjem izlaganju akcenat je bio na direktnom inženjeringu (direct engineering) po kojem programeri imaju ogromnu uštedu u vremenu u koliko im je dizajner baze podataka dostavio gotovu šemu baze podataka na odgovarajućem serveru sa kojim Java EE aplikacija treba da se poveže.

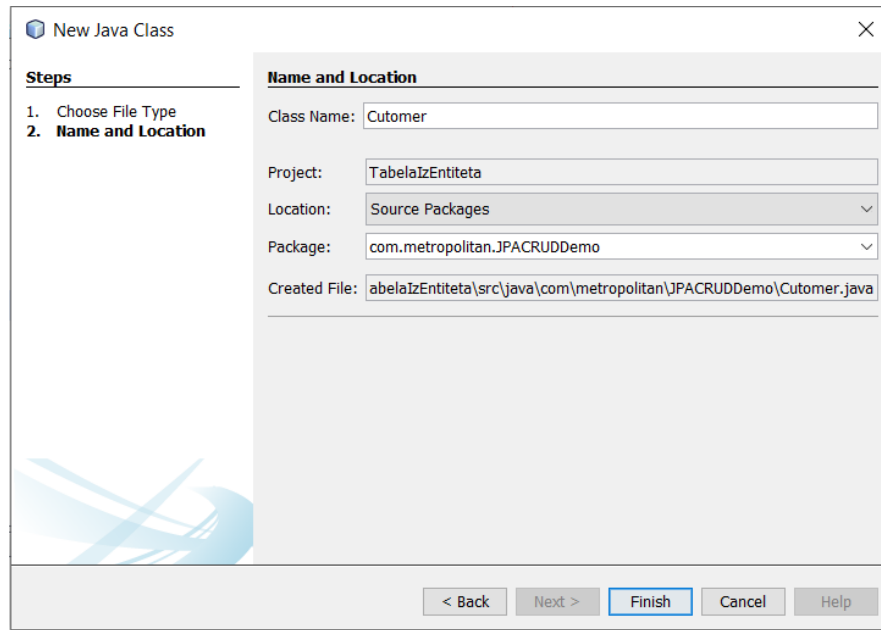
Kod inverznog / obrnutog inženjeringa (reverse engineering), programer se ponaša drugačije. On kreira entitetske klase, na osnovu zahteva, a zatim iz njih, primenom JPA alata, kreira odgovarajuće tabele sa kojima se mapiraju entiteti. U ovom slučaju nema demonstrirane uštede u vremenu jer je programer prinuđen da manuelno kreira svaki entitet.

Po već utvrđenoj praksi kompletno izlaganje će teći preko adekvatnog primera. Zahtev je sledeći:

1. Kreirajte CRUD JEE aplikaciju;
2. Aplikacija sadrži jedan entitet Customer (id, firstname, lastname);
3. Koristite JPA za ORM;
4. Ne postoji baza podataka - programer će sam kreirati tabele nakon programiranja entiteta;
5. Iz osobina entiteta će biti kreirane kolone tabele baze podataka na sledeći način:

- *id* je celobrojna vrednost i primarni ključ;
- *firstname* i *lastname* su stringovi.

Programer pristupa kreiranju POJO klase Customer (slika 1).



Slika 7.1 Početak kreiranja entiteta [izvor: autor]

## KREIRANJE ENTITETA

*Programer obeležava kreiranu klasu anotacijom `Entity`.*

Programer pristupa kreiranju POJO klase `Customer`, koju će da obeleži anotacijom `@Entity`, a zatim i anotacijom `@Table(name = "customer")`. Na ovaj način je kreirana je entitetska klasa `Customer` koja će u budućnosti da se mapira sa tabelom `customer` iz odgovarajuće baze podataka.

Programer dalje pristupa kreiranju privatnih polja:

- `id` tipa `long`;
- `firstname` tipa `String`;
- `lastname` tipa `String`;

Odmah nakon obavljenog navedenog zadatka kreira konstruktore, `set` i `get` metode za kreirana privatna polja.

```
@Entity
@Table(name = "customer")
public class Customer {
    private Long id;
    private String firstname;
    private String lastname;
    public Customer() {
    }

    public Customer(Long id) {
        this.id = id;
    }
}
```

```

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}

public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}
}

```

Programer ima još malo posla, dodaće anotaciju `@XmlRootElement` koja omogućava da se objekti ove klase reprezentuju kao XML ili JSON stringovi. Konačno dodaje se i malo JPA upita koji će obezbediti izvođenje **CRUD** operacija nad objektima entiteta, odnosno nad odgovarajućom tabelom baze podataka.

Definicija entitetske klase je završena dodavanjem metoda `hashCode()`, `equals()` i `toString()` koje su poprilično opšte i neće biti detaljno diskutovane. Sledi listing kreiranog entiteta. Konačno, **svaki entitet mora da bude serijalizovan**.

```

@Entity
@Table(name = "customer")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Customer.findAll", query = "SELECT c FROM Customer c")
    , @NamedQuery(name = "Customer.findById", query = "SELECT c FROM Customer c WHERE c.id = :id")
    , @NamedQuery(name = "Customer.findByFirstname", query = "SELECT c FROM Customer c WHERE c.firstname = :firstname")
    , @NamedQuery(name = "Customer.findByLastname", query = "SELECT c FROM Customer c WHERE c.lastname = :lastname")})
public class Customer implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)

```

```
@NotNull
@Column(name = "ID")
private Long id;
@Size(max = 255)
@Column(name = "FIRSTNAME")
private String firstname;
@Size(max = 255)
@Column(name = "LASTNAME")
private String lastname;

public Customer() {
}

public Customer(Long id) {
    this.id = id;
}

public Long getId() {
    return id;
}

public void setId(Long id) {
    this.id = id;
}

public String getFirstname() {
    return firstname;
}

public void setFirstname(String firstname) {
    this.firstname = firstname;
}

public String getLastname() {
    return lastname;
}

public void setLastname(String lastname) {
    this.lastname = lastname;
}

@Override
public int hashCode() {
    int hash = 0;
    hash += (id != null ? id.hashCode() : 0);
    return hash;
}

@Override
public boolean equals(Object object) {
    // TODO: Warning - this method won't work in the case the id fields are not
set
    if (!(object instanceof Customer)) {
```

```

        return false;
    }
    Customer other = (Customer) object;
    if ((this.id == null && other.id != null) || (this.id != null &&
!this.id.equals(other.id))) {
        return false;
    }
    return true;
}

@Override
public String toString() {
    return "com.metropolitan.JPACRUDDemo.Customer[ id=" + id + " ]";
}
}

```

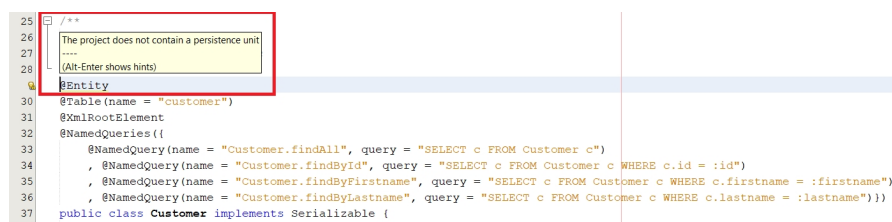
## KREIRANJE DATOTEKE JEDINICE PERZISTENCIJE

*Postojanje entitetske klase u veb aplikaciji je nemoguće bez datoteke jedinice perzistencije.*

Postojanje entitetske klase u veb aplikaciji je nemoguće bez datoteke jedinice perzistencije (persistence unit). Preko ove datoteke je definisano koje će klase biti mapirane u tabele, a definišu se i odgovarajuće strategije mapiranja:

- **none** - nema modifikacija u bazi podataka;
- **create** - kreirati tabelu u bazi podataka za entitet za koji ne postoji odgovarajuća tabela, vrši se ažuriranje šeme baze podataka;
- **drop and create** - svaki put kada se pokrene aplikacija kreira se nova šema baze podataka, a prethodna se briše zajedno sa podacima.

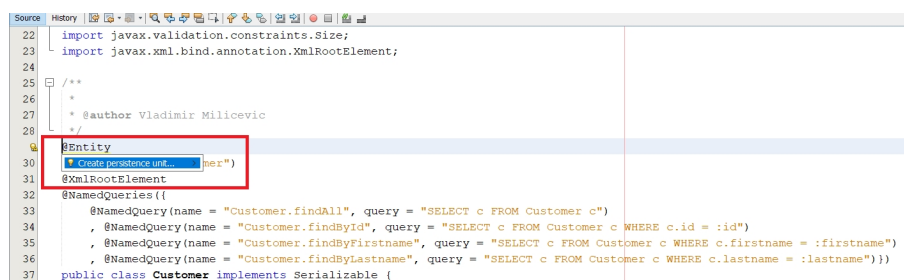
Savremena razvojna okruženja (NetBeans, Eclipse, IntelliJ) veoma lako prepoznaju da u projektu postoji entitet ali da nije kreirana jedinica perzistencije (slika 2).



Slika 7.2 Nepostojanje jedinice perzistencije u projektu [izvor: autor]

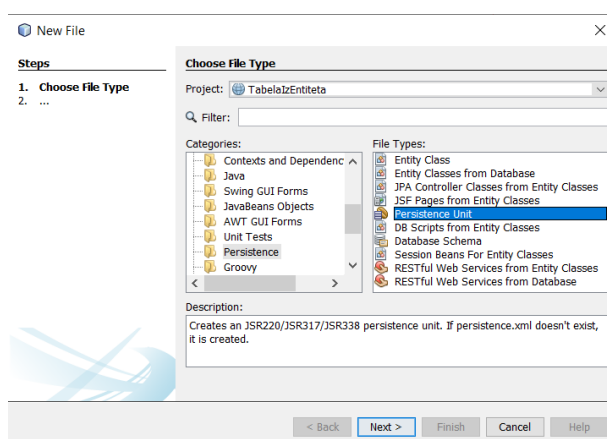
Takođe, razvojno okruženje može da ponudi programeru prečicu za kreiranje jedinice perzistencije (slika 3)





Slika 7.3 Prečica za kreiranje jedinice perzistencije [izvor: autor]

Konačno, programer može ceo proces kreiranja ove datoteke da prođe i na standardni način kreiranjem novog fajla iz kategorije *Persistence*, tipa *Persistence unit* (slika 4).

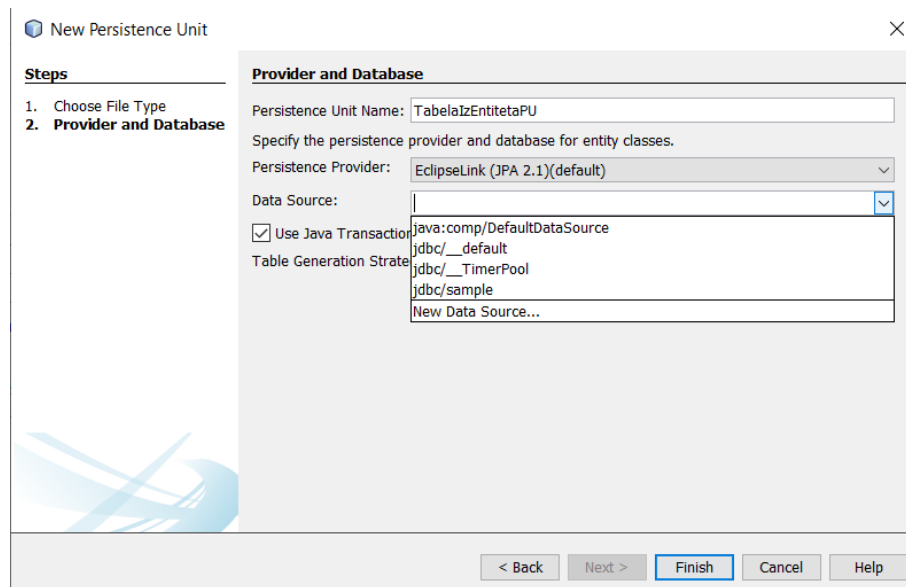


Slika 7.4 Kreiranje nove jedinice prezistencije [izvor: autor]

## DEFINISANJE IZVORA PODATAKA

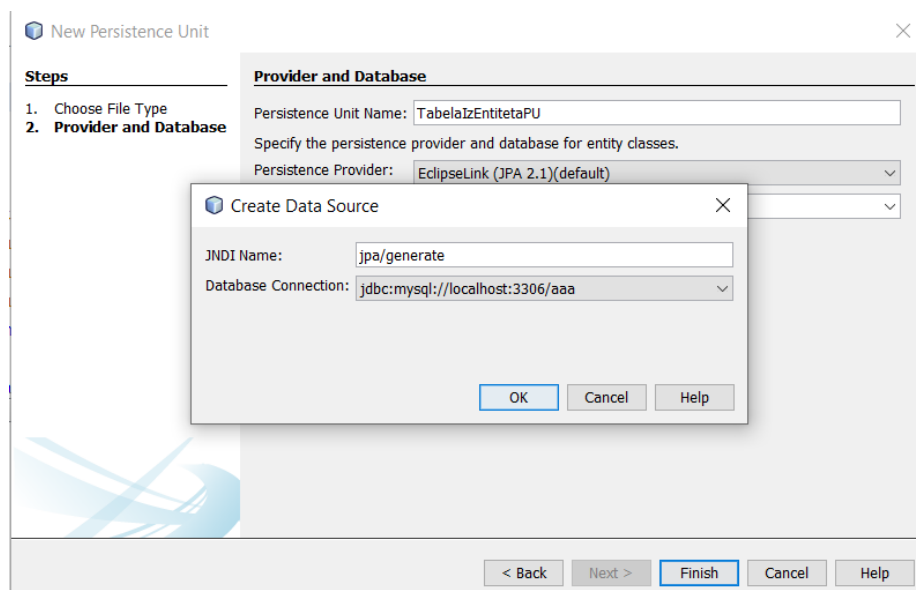
*Pre kodiranja programer dodeljuje naziv datoteci, a zatim bira izvor podataka.*

U sledećem koraku programer dodeljuje naziv datoteci, a zatim bira ili kreira nov izvor podataka (data source), odnosno bazu podataka sa kojom će aplikacija biti povezana (slika 5)



Slika 7.5 Kreiranje naziva za PU i izbor izvora podataka [izvor: autor]

Sledećom slikom je prikazano definisanje JNDI naziva i izbora baze podataka sa kojom će aplikacija biti povezana.



Slika 7.6 JNDI naziv i izvor podataka [izvor: autor]

## IZBOR JPA PROVAJDERA I STRATEGIJE MAPIRANJA

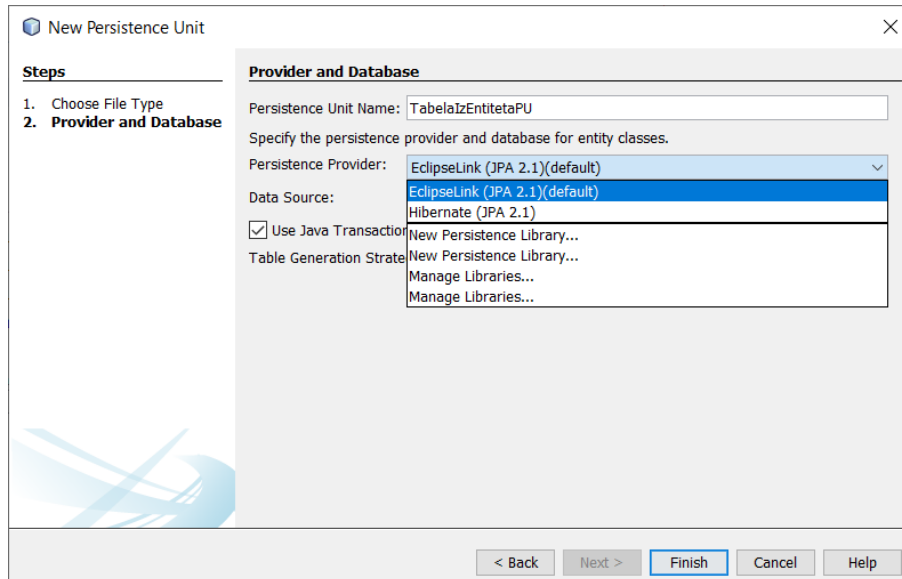
*Izbor JPA provajdera i strategije mapiranja je poslednji korak pre konačne definicije za PU.*

U zavisnosti kako želi da implementira JPA, programer može da se opredeli za odgovarajućeg provajdera:

- EclipseLink - podrazumevani;

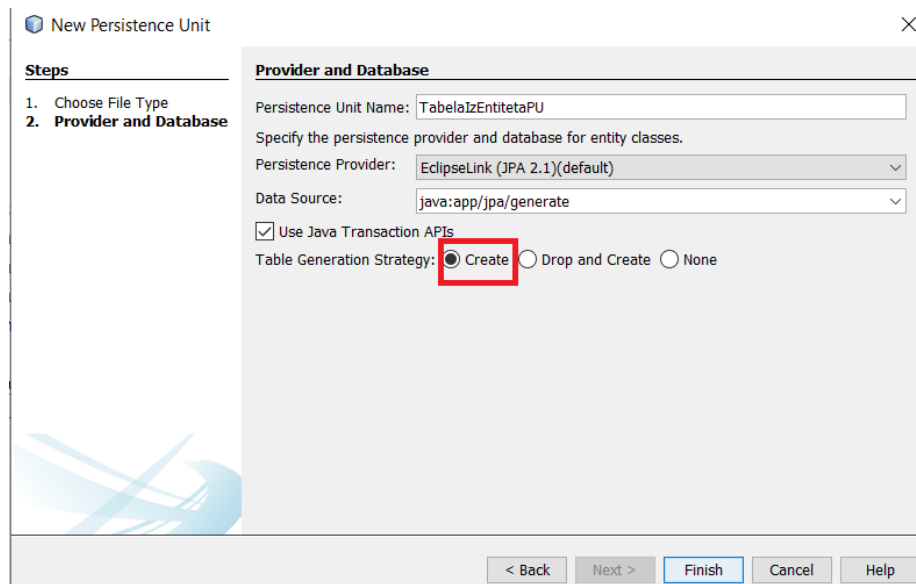
- **Hibernate** - alternativni provajder.

U ovom slučaju, izbor će pasti na *EclipseLink*, a sa *Hibernate* implementacijom ćemo se više baviti u IT355 - Veb sistemi 2.



Slika 7.7 Izbor JPA provajdera [izvor: autor]

Konačno, programer bira strategiju mapiranja *create*, a to znači da će za svaki entitet biti kreirana nova tabela u bazi podataka, ako pre toga nije postojala.



Slika 7.8 Izbor strategije mapiranja [izvor: autor]

Sada je sve spremno za kodiranje jedinice perzistencije.

# KODIRANJE JEDINICE PERZISTENCIJE

*Kodiranje jedinice perzistencije je poslednji korak pre kreiranja tabela baze podataka.*

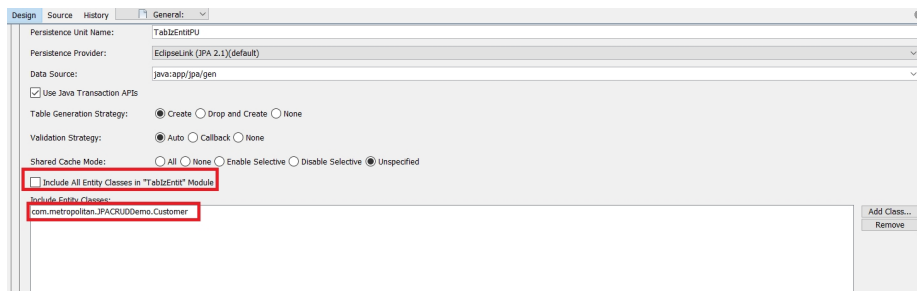
Rezultat izvođenja prethodnih aktivnosti jeste kreiranje dve datoteke:

- [glassfish-resources.xml](#) - koja definiše pravila za povezivanje sa bazom podataka (izvor podataka);
- [persistence.xml](#) - datoteka jedinice perzistencije;

Sledi listing datoteke [glassfish-resources.xml](#):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE resources PUBLIC "-//GlassFish.org//DTD GlassFish Application Server 3.1
Resource Definitions//EN" "http://glassfish.org/dtds/glassfish-resources_1_5.dtd">
<resources>
  <jdbc-connection-pool allow-non-component-callers="false"
associate-with-thread="false" connection-creation-retry-attempts="0"
connection-creation-retry-interval-in-seconds="10" connection-leak-reclaim="false"
connection-leak-timeout-in-seconds="0" connection-validation-method="auto-commit"
datasource-classname="com.mysql.jdbc.jdbc2.optional.MysqlDataSource"
fail-all-connections="false" idle-timeout-in-seconds="300"
is-connection-validation-required="false" is-isolation-level-guaranteed="true"
lazy-connection-association="false" lazy-connection-enlistment="false"
match-connections="false" max-connection-usage-count="0" max-pool-size="32"
max-wait-time-in-millis="60000" name="mysql_aaa_rootPool"
non-transactional-connections="false" pool-resize-quantity="2"
res-type="javax.sql.DataSource" statement-timeout-in-seconds="-1"
steady-pool-size="8" validate-atmost-once-period-in-seconds="0"
wrap-jdbc-objects="false">
    <property name="serverName" value="localhost"/>
    <property name="portNumber" value="3306"/>
    <property name="databaseName" value="aaa"/>
    <property name="User" value="root"/>
    <property name="Password" value=""/>
    <property name="URL" value="jdbc:mysql://localhost:3306/
aaa?zeroDateTimeBehavior=convertToNull"/>
    <property name="driverClass" value="com.mysql.jdbc.Driver"/>
  </jdbc-connection-pool>
  <jdbc-resource enabled="true" jndi-name="java:app/jpa/gen" object-type="user"
pool-name="mysql_aaa_rootPool"/>
</resources>
```

Kodiranje jedinice perzistencije je poslednji korak pre kreiranja tabela baze podataka. Savremena razvojna okruženja omogućavaju da se preko čarobnjaka (wizard) dodaju odgovarajuće osobine koje će biti umetnute u xml kod jedinice perzistencije. Ko je veštiji u radu sa XML kodom, to može da uradi i manuelno.



Slika 7.9 Izbor entiteta za mapiranje [izvor: autor]

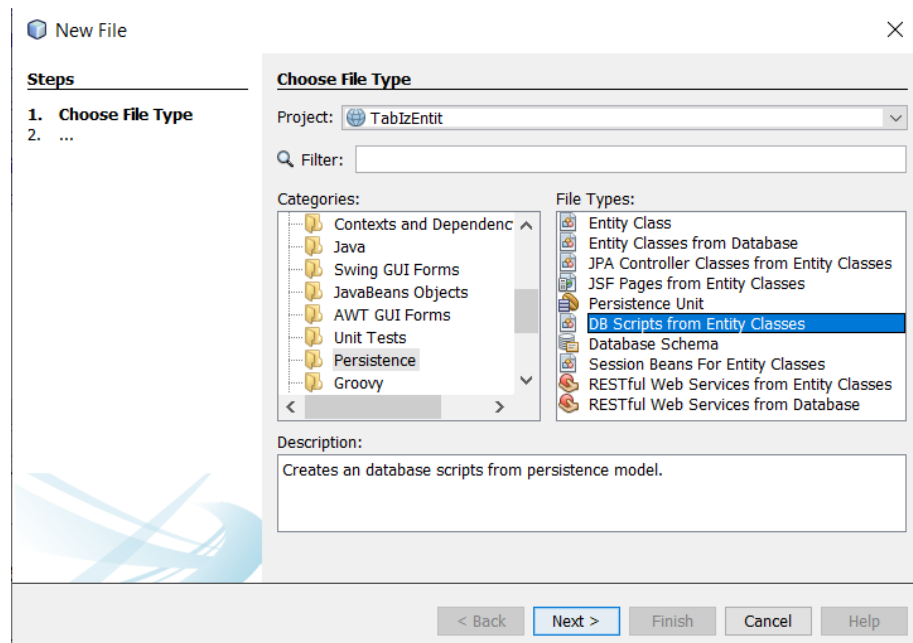
Izborom klasa koje se mapiraju u tabele, završena je definicija datoteke *persistence.xml* i njen sadržaj odgovara sledećem listingu:

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" 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_1.xsd">
  <persistence-unit name="TabIzEntitPU" transaction-type="JTA">
    <jta-data-source>java:app/jpa/gen</jta-data-source>
    <class>com.metropolitan.JPACRUDDemo.Customer</class>
    <exclude-unlisted-classes>true</exclude-unlisted-classes>
    <properties>
      <property name="javax.persistence.schema-generation.database.action"
value="create"/>
    </properties>
  </persistence-unit>
</persistence>
```

## KREIRANJE GENERIŠUĆEG SQL SKRIPTA

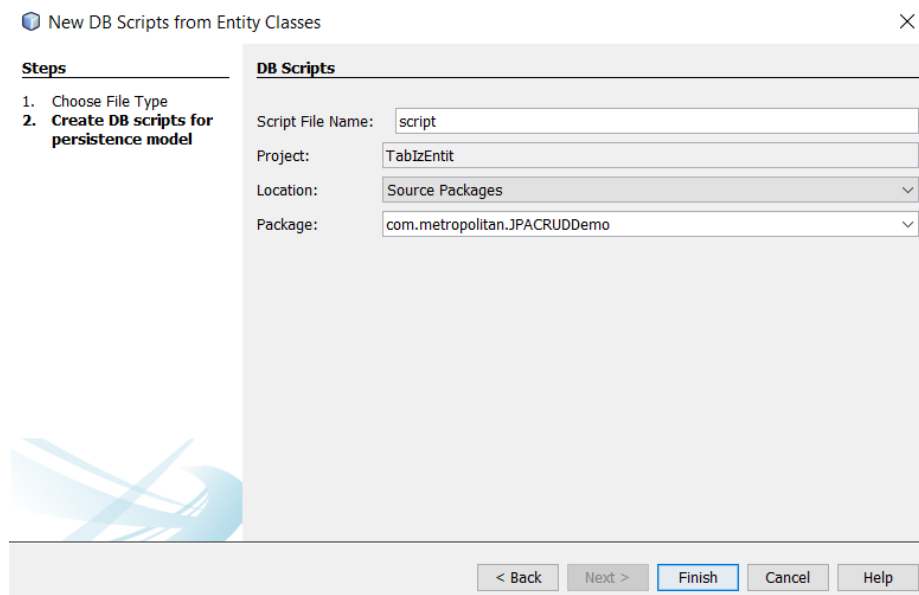
*Nakon kodiranja jedinice perzistencije pristupa se kreiranju SQL skripta za kreiranje tabela BP.*

Nakon kodiranja jedinice perzistencije pristupa se kreiranju SQL skripta za kreiranje tabela baze podataka. Procedura je veoma jednostavna. Kreira se nova datoteka iz kategorije *Persistence*, tipa *DB Scripts From Entity Classes*. Navedeno je prikazano sledećom slikom:



Slika 7.10 Kreiranje generišućeg SQL skripta [izvor: autor]

Na osnovu kreirane jedinice perzistencije identifikuju se klase iz kojih će biti kreirane odgovarajuće tabele i jedino što je potrebno jeste da se dodeli naziv ovoj SQL datoteci (sledeća slika).



Slika 7.11 Dodela naziva za SQL skript za generisanje tabela baze podataka [izvor: autor]

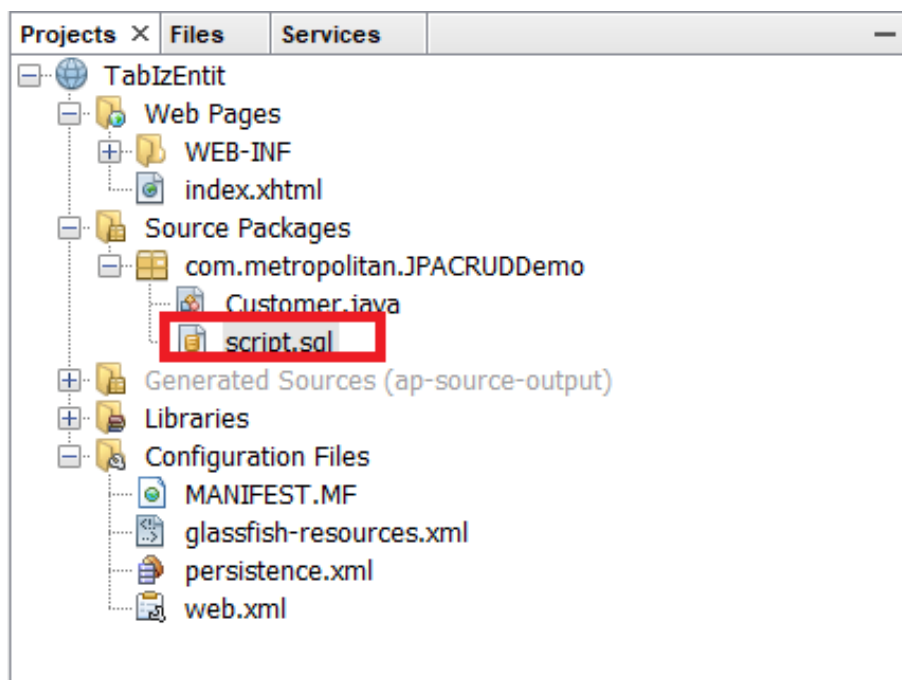
Sledećim listingom je prikazan sadržaj kreirane datoteke.

```
CREATE TABLE customer (ID BIGINT NOT NULL, FIRSTNAME VARCHAR(255), LASTNAME
VARCHAR(255), PRIMARY KEY (ID))
```

## KREIRANJE TABELA BAZE PODATAKA

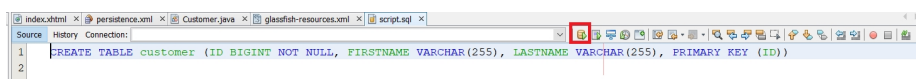
*Izvršavanjem kreiranog SQL skripta obavlja se popunjavanje baze podataka tabelama.*

Kreirani skript se nalazi sada na odgovarajućem mestu u projektu (sledeća slika).



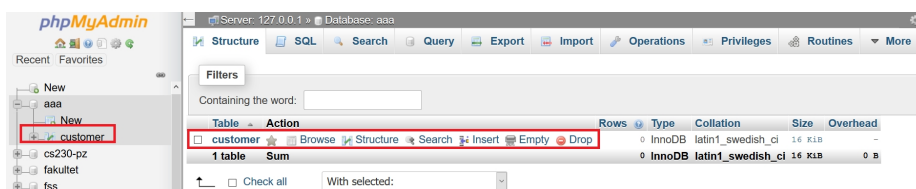
Slika 7.12 Položaj SQL skripa u projektu [izvor: autor]

Izvršavanjem kreiranog SQL skripta obavlja se popunjavanje baze podataka odgovarajućim tabelama. U razvojnom okruženju se pokreće skript na način prikazan sledećom slikom:



Slika 7.13 Pokretanje kreiranog SQL skripta (Izvor: autor)

Da je sve uspešno obavljeno, moguće je proveriti u aplikaciji za upravljanje bazom podataka. Zaista, odgovarajuća tabela je kreirana, a to je prikazano sledećom slikom:



Slika 7.14 Kreirana tabela u bazi podataka [izvor: autor]

Sada je kreiran inicijalni backend i programiranje aplikacije može da teče dalje u smeru dodavanja novih specifičnosti i funkcionalnosti.

## ▼ Poglavlje 8

# Kreiranje JSF aplikacija iz JPA entiteta

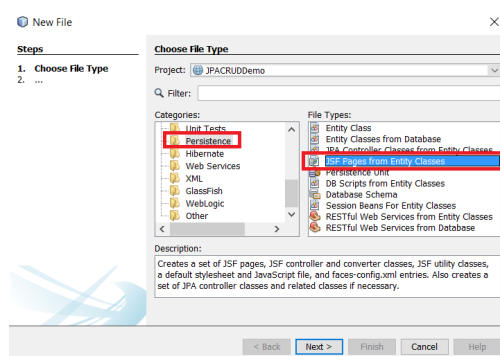
## KREIRANJE JSF DATOTEKA

*Kreiranjem JSF datoteka kompletira se veb aplikacija.*

Budući da su u prethodnom radu kreirani JPA entiteti, koji predstavljaju objektno - orijentisanu implementaciju tabela baze podataka, a zatim su kreirani i odgovarajuće DAO datoteke za upravljanje podacima u okviru aplikacije, neophodno je razviti mehanizme pomoću kojih će korisnim moći da interaguje sa aplikacijom.

U aktuelnom primeru, primenom razvojnog alata NetBeans IDE, biće pokazano kako je moguće automatski kreirati JSF stranice i na taj način učiniti veb aplikaciju koja se razvija kompletno funkcionalnom.

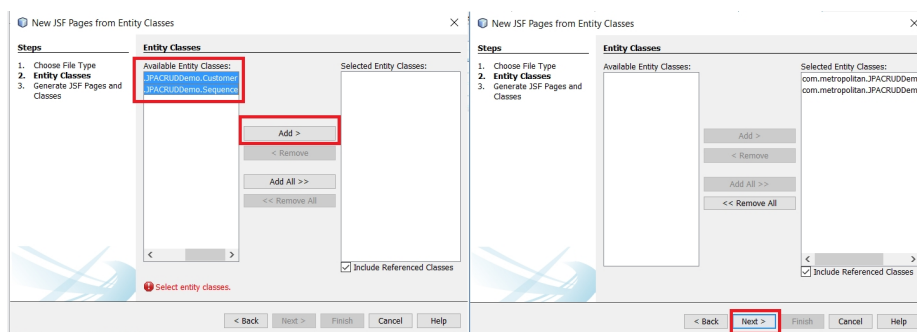
U razvojnom okruženju, za aktuelni projekat, bira se opcija **File**, a zatim i **New File**. Otvara se prozor **New File** u kojem se bira kategorija datoteke (u ovom slučaju **JavaServer Faces Persistence**) i tip datoteke (u ovom slučaju **JSF Pages from Entity Classes**) kao na sledećoj slici.



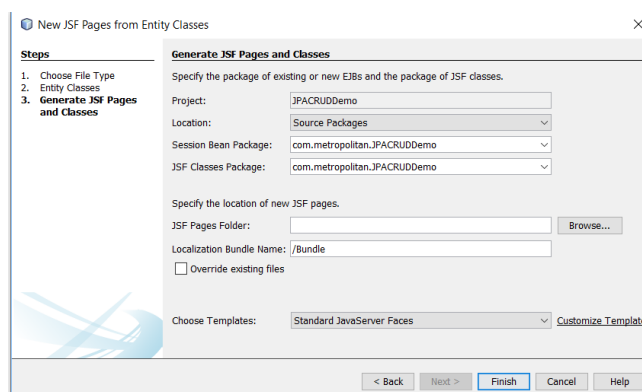
Slika 8.1 Kreiranje novih JSF Pages from Entity Classes datoteka [izvor: autor]

Klikom na dugme **Next** otvara se prozor za izbor klasa JPA entiteta. Klase se markiraju i klikom na dugme **Add** dodaju za dalji proces kreiranja JSF stranica (Slika2). Klikom da **Next** odlazi se u prozor gde se završava ovaj proces kreiranja JSF stranica (Slika3).





Slika 8.2 Izbor JPA entiteta za generisanje JSF datoteka [izvor: autor]



Slika 8.3 Kreiranje JSF stranica - završni korak [izvor: autor]

## POČETNA STRANICA

*Prolaskom kroz prikazani čarobnjak kreirane su JSF stranice.*

Prolaskom kroz prikazani čarobnjak kreirane su JSF stranice za sve klase entiteta koje su bile izabrane. Kreirana je i **index.xhtml** stranica od koje počinje navigacija kroz kreiranu veb aplikaciju. Listing stranice **index.xhtml** je priložen:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Facelet Title</title>
    <h:outputStylesheet name="css/jsfcrud.css"/>
  </h:head>
  <h:body>
    Hello from Facelets

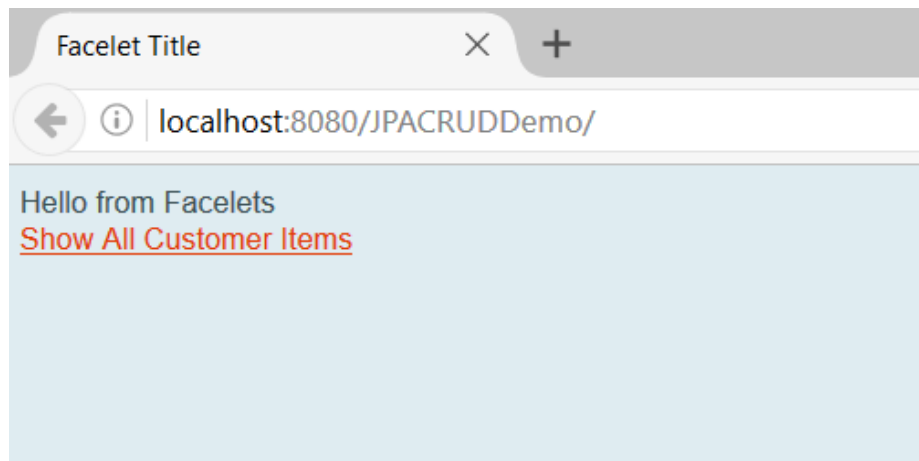
    <h:link outcome="/customer/List" value="Show All Customer Items"/>
  </h:body>
```

```
</html>
```

Na početnoj stranici definisan je link kojim se vrši navigacija ka stranici `List.xhtml`, koja je generisana u folderu `customer` i koja je nastala iz JPA klase `Customer`. Još neke datoteke su na isti način našle svoje mesto u ovom folderu, a to su: `Create.xhtml`, `Edit.xhtml` i `View.xhtml`.

Stranicom `List.xhtml` se prikazuje lista dostupnih korisnika iz baze podataka; `Create.xhtml` kreira novog korisnika; `Edit.xhtml` dozvoljava da se podaci o korisniku ažuriraju i `View.xhtml` prikazuje podatke o korisniku.

Zbog jednostavnosti, napravljena je prosta veb aplikacija čije JSF stranice se odnose na JPA klasu `Customer.java`. Pokretanjem aplikacije, učitava se stranica `index.xhtml` na način prikazan sledećom slikom:

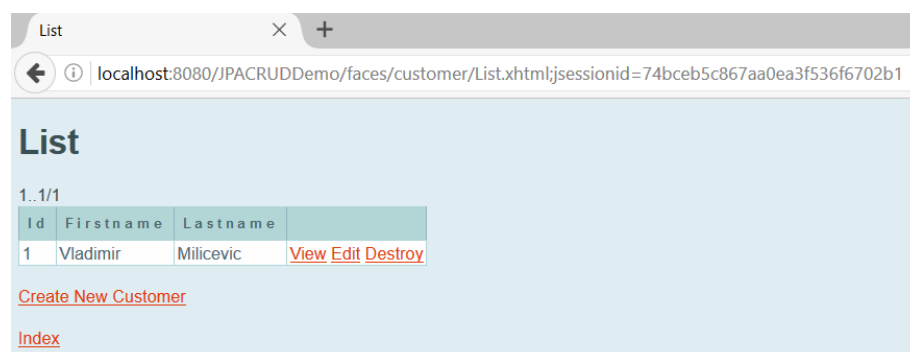


Slika 8.4 Automatski generisan JSF stranica `index.xhtml` [izvor: autor]

## DATOTEKA LIST.XHTML

*Demonstracija ostalih automatski generisanih JSF datoteka iz JPA klase entiteta.*

Klikom na link `Show All Customer Items`, na početnoj stranici, vrši se navigacija do sledeće automatski generisane stranice `List.xhtml`. Ovom stranicom se prikazuje lista svih korisnika koji su dostupni u bazi podataka (sledeća slika).



Slika 8.5 Lista korisnika iz baze podataka [izvor: autor]

Kada se pogleda priložena stranica primećuju se linkovi ka ostalim stranicama, pored prikazane liste korisnika. Moguće je kreirati novog korisnika i dodati ga u bazu podataka (stranica [Create.xhtml](#)) klikom na link [Create New Customer](#). Moguće je, klikom na link [View](#), otići na stranicu sa istim nazivom i pogledati podatke za konkretnog korisnika. Takođe, klikom na link [Edit](#), omogućen je odlazak na stranicu za ažuriranje korisničkih podataka.

Na kraju, moguće je brisanje korisnika ([Destroy](#)) i povratak na početnu stranicu ([Index](#)).

Sledećim listingom priložen je JSF kod stranice [List.xhtml](#).

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:p="http://primefaces.org/ui">

  <ui:composition template="/template.xhtml">

    <ui:define name="title">
      <h:outputText value="#{bundle.ListCustomerTitle}"></h:outputText>
    </ui:define>

    <ui:define name="body">
      <h:form id="CustomerListForm">
        <p:panel header="#{bundle.ListCustomerTitle}">
          <p:dataTable id="datalist" value="#{customerController.items}"
            var="item"
            selectionMode="single"
            selection="#{customerController.selected}"
            paginator="true"
            rowKey="#{item.id}"
            rows="10"
            rowsPerPageTemplate="10,20,30,40,50"
            >

            <p:ajax event="rowSelect" update="createButton viewButton
editButton deleteButton"/>
            <p:ajax event="rowUnselect" update="createButton viewButton
editButton deleteButton"/>

            <p:column>
              <f:facet name="header">
                <h:outputText
value="#{bundle.ListCustomerTitle_firstName}">
              </f:facet>
              <h:outputText value="#{item.firstName}">
            </p:column>
```

```

        <p:column>
            <f:facet name="header">
                <h:outputText
value="#{bundle.ListCustomerTitle_lastName}"/>
            </f:facet>
            <h:outputText value="#{item.lastName}"/>
        </p:column>
        <p:column>
            <f:facet name="header">
                <h:outputText
value="#{bundle.ListCustomerTitle_id}"/>
            </f:facet>
            <h:outputText value="#{item.id}"/>
        </p:column>
        <f:facet name="footer">
            <p:commandButton id="createButton"
icon="ui-icon-plus" value="#{bundle.Create}"
actionListener="#{customerController.prepareCreate}" update=":CustomerCreateForm"
oncomplete="PF('CustomerCreateDialog').show()"/>
            <p:commandButton id="viewButton"
icon="ui-icon-search" value="#{bundle.View}" update=":CustomerViewForm"
oncomplete="PF('CustomerViewDialog').show()" disabled="#{empty
customerController.selected}"/>
            <p:commandButton id="editButton"
icon="ui-icon-pencil" value="#{bundle.Edit}" update=":CustomerEditForm"
oncomplete="PF('CustomerEditDialog').show()" disabled="#{empty
customerController.selected}"/>
            <p:commandButton id="deleteButton"
icon="ui-icon-trash" value="#{bundle.Delete}"
actionListener="#{customerController.destroy}" update=":growl,datalist"
disabled="#{empty customerController.selected}"/>
        </f:facet>
    </p:dataTable>
</p:panel>
</h:form>

    <ui:include src="Create.xhtml"/>
    <ui:include src="Edit.xhtml"/>
    <ui:include src="View.xhtml"/>
</ui:define>
</ui:composition>

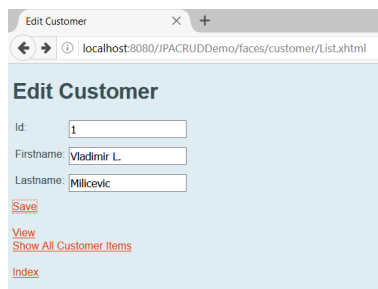
</html>

```

## DATOTEKA EDIT.XHTML

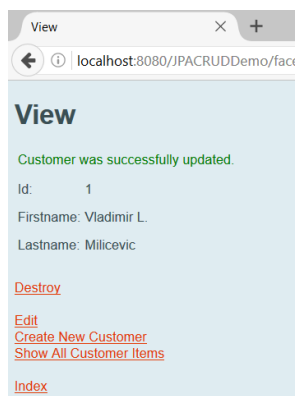
*Ovom stranicom je omogućeno ažuriranje podataka, iz baze podataka, o korisniku.*

U prethodnom izlaganju je opisan način kako se stiže do stranice za ažuriranje kosničkih podataka koja je prikazana sledećom slikom.



Slika 8.6 Ažuriranje korisničkih podataka [izvor: autor]

Kada se izvesne informacije promene, klikom na link **Save** one će biti zapamćene (sledeća slika)



Slika 8.7 Informacija o ažuriranju podataka [izvor: autor]

Kao što je moguće videti i ova stranica dozvoljava navigaciju ka početnoj stranici, ali i stranicama za pregled korisnikovih podataka, kao i liste svih korisnika. Sledi listing stranice **Edit.xhtml**.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">

  <ui:composition template="/template.xhtml">
    <ui:define name="title">
      <h:outputText value="#{bundle.EditCustomerTitle}"></h:outputText>
    </ui:define>
    <ui:define name="body">
      <h:panelGroup id="messagePanel" layout="block">
        <h:messages errorStyle="color: red" infoStyle="color: green"
          layout="table"/>
      </h:panelGroup>
      <h:form>
```

```

        <h:panelGrid columns="2">
            <h:outputLabel value="#{bundle.EditCustomerLabel_id}" for="id"
/>
            <h:inputText id="id" value="#{customerController.selected.id}"
title="#{bundle.EditCustomerTitle_id}" required="true"
requiredMessage="#{bundle.EditCustomerRequiredMessage_id}"/>
            <h:outputLabel value="#{bundle.EditCustomerLabel_firstname}"
for="firstname" />
            <h:inputText id="firstname"
value="#{customerController.selected.firstname}"
title="#{bundle.EditCustomerTitle_firstname}" />
            <h:outputLabel value="#{bundle.EditCustomerLabel_lastname}"
for="lastname" />
            <h:inputText id="lastname"
value="#{customerController.selected.lastname}"
title="#{bundle.EditCustomerTitle_lastname}" />
        </h:panelGrid>
        <h:commandLink action="#{customerController.update}"
value="#{bundle.EditCustomerSaveLink}"/>
        <br/>
        <br/>
        <h:link outcome="View" value="#{bundle.EditCustomerViewLink}"/>
        <br/>
        <h:commandLink action="#{customerController.prepareList}"
value="#{bundle.EditCustomerShowAllLink}" immediate="true"/>
        <br/>
        <br/>
        <h:link outcome="/index" value="#{bundle.EditCustomerIndexLink}" />
    </h:form>
</ui:define>
</ui:composition>

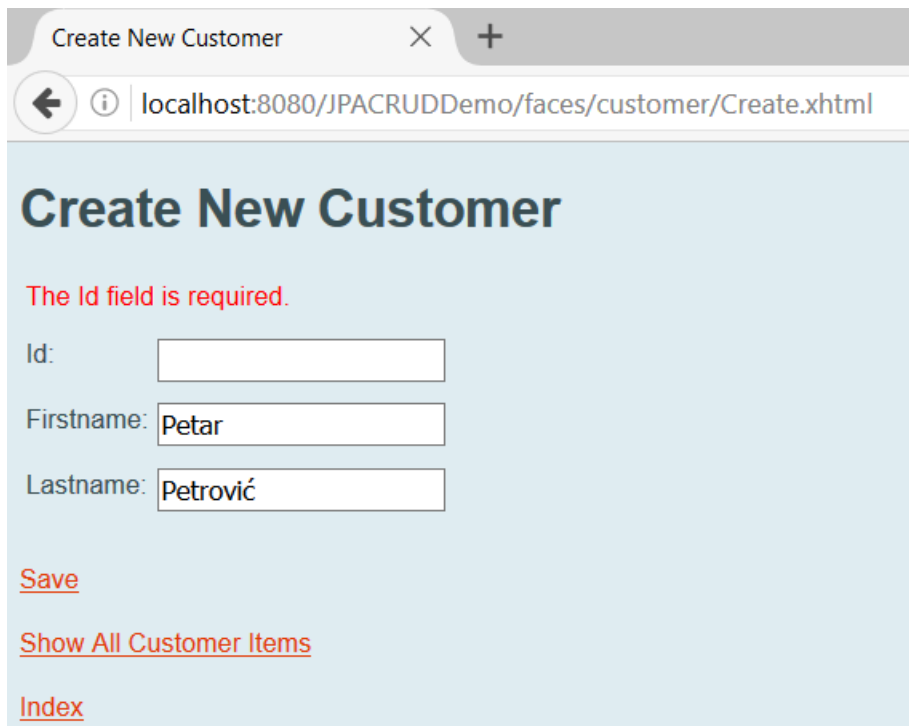
</html>

```

## DATOTEKA CREATE.XHTML

*Automatski je generisana i JSF stranica za kreiranje i unos novih korisnika u bazu podataka.*

Automatski je generisana i JSF stranica za kreiranje i unos novih korisnika u bazu podataka, koja je dobila naziv **Create.xhtml** i prikazana je sledećom slikom.



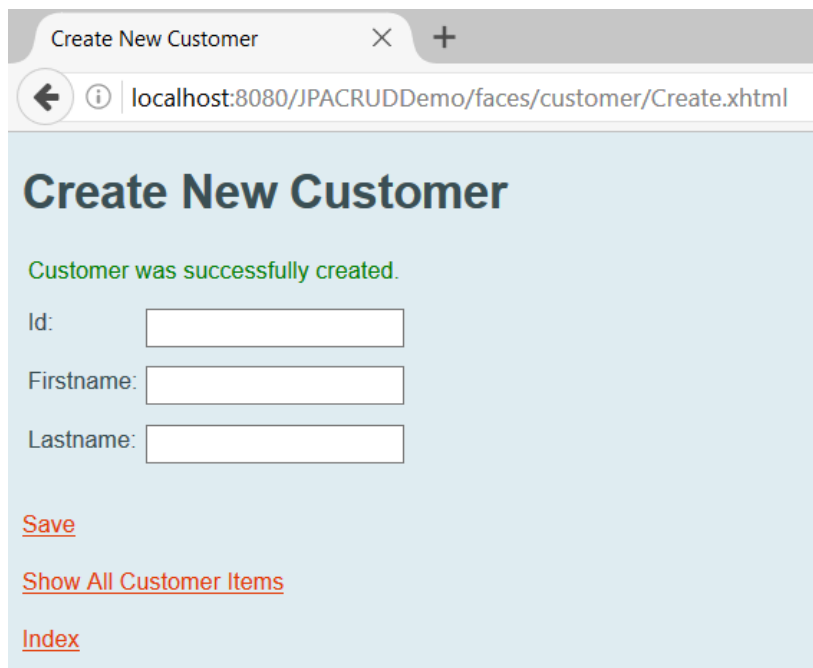
The screenshot shows a web browser window with the title 'Create New Customer'. The address bar displays 'localhost:8080/JPACRUDDemo/faces/customer/Create.xhtml'. The main heading is 'Create New Customer'. Below the heading, a red error message states 'The Id field is required.' The form contains three input fields: 'Id:' (empty), 'Firstname:' (containing 'Petar'), and 'Lastname:' (containing 'Petrović'). At the bottom of the form, there are three links: 'Save', 'Show All Customer Items', and 'Index'.

Slika 8.8 Kreiranje novog korisnika [izvor: autor]

Na slici je moguće primetiti da je sva polja, koja su obavezna, neophodno popuniti inače se javlja poruka sa greškom.

Takođe, stranica obuhvata navigaciju ka početnoj stranici i ka stranici koja prikazuje listu svih postojećih korisnika iz baze podataka. Link **Save** realizuje snimanje unetih podataka o novom korisniku.

Forma sa Slike 8, korektno je popunjena, izvršeno je snimanje korisnika i dobijena je informacija da je korisnik uspešno sačuvan u bazi podataka.



The screenshot shows the same web browser window as before, but the error message has been replaced by a green success message: 'Customer was successfully created.' The input fields are now empty: 'Id:', 'Firstname:', and 'Lastname:'. The links 'Save', 'Show All Customer Items', and 'Index' remain at the bottom of the form.

Slika 8.9 Dodati je nov korisnik [izvor: autor]

## KREIRANJE NOVOG KORISNIKA

*Ponovno učitavanje stranice List.xhtml za prikazivanje ažurirane liste korisnika.*

Prethodno prikazana stranica **Create.xhtml** realizovana je kodom koji je priložen sledećim listingom.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core"
      xmlns:p="http://primefaces.org/ui">

  <ui:composition template="/template.xhtml">

    <ui:define name="title">
      <h:outputText value="#{bundle.ListCustomerTitle}"></h:outputText>
    </ui:define>

    <ui:define name="body">
      <h:form id="CustomerListForm">
        <p:panel header="#{bundle.ListCustomerTitle}">
          <p:dataTable id="datalist" value="#{customerController.items}"
            var="item"
              selectionMode="single"
            selection="#{customerController.selected}"
              paginator="true"
              rowKey="#{item.id}"
              rows="10"
              rowsPerPageTemplate="10,20,30,40,50"
            >

            <p:ajax event="rowSelect" update="createButton viewButton
            editButton deleteButton"/>
            <p:ajax event="rowUnselect" update="createButton viewButton
            editButton deleteButton"/>

            <p:column>
              <f:facet name="header">
                <h:outputText
            value="#{bundle.ListCustomerTitle_firstName}" />
              </f:facet>
              <h:outputText value="#{item.firstName}" />
            </p:column>
```



```

        <p:column>
            <f:facet name="header">
                <h:outputText
value="#{bundle.ListCustomerTitle_lastName}"/>
            </f:facet>
            <h:outputText value="#{item.lastName}"/>
        </p:column>
        <p:column>
            <f:facet name="header">
                <h:outputText
value="#{bundle.ListCustomerTitle_id}"/>
            </f:facet>
            <h:outputText value="#{item.id}"/>
        </p:column>
        <f:facet name="footer">
            <p:commandButton id="createButton"
icon="ui-icon-plus" value="#{bundle.Create}"
actionListener="#{customerController.prepareCreate}" update=":CustomerCreateForm"
oncomplete="PF('CustomerCreateDialog').show()"/>
            <p:commandButton id="viewButton"
icon="ui-icon-search" value="#{bundle.View}" update=":CustomerViewForm"
oncomplete="PF('CustomerViewDialog').show()" disabled="#{empty
customerController.selected}"/>
            <p:commandButton id="editButton"
icon="ui-icon-pencil" value="#{bundle.Edit}" update=":CustomerEditForm"
oncomplete="PF('CustomerEditDialog').show()" disabled="#{empty
customerController.selected}"/>
            <p:commandButton id="deleteButton"
icon="ui-icon-trash" value="#{bundle.Delete}"
actionListener="#{customerController.destroy}" update=":growl,datalist"
disabled="#{empty customerController.selected}"/>
        </f:facet>
    </p:dataTable>
</p:panel>
</h:form>

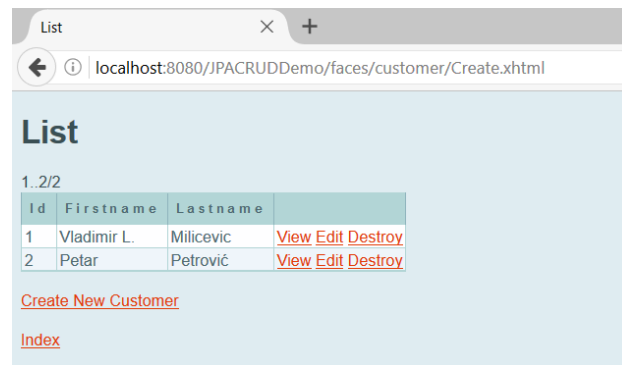
    <ui:include src="Create.xhtml"/>
    <ui:include src="Edit.xhtml"/>
    <ui:include src="View.xhtml"/>
</ui:define>
</ui:composition>

</html>

```

Kao što je moguće primetiti odavde je moguće ponovna navigacija ka stranici koja prikazuje listu korisnika iz baze podataka. Klikom na link **Show All Customer Items** još jednom će se proveriti funkcionalnost aplikacije i zaokružiti izlaganje o bazama podataka u JPA veb aplikacijama.

Ažurirana lista korisnika prikazana je sledećom slikom.



Slika 8.10 Ažurirana lista korisnika na stranici List.xhtml. [izvor: autor]

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## ▼ Poglavlje 9

# Pokazni primer - JPA

## POSTAVKA (45 MIN)

*Kreira se veb aplikacija nad bazom podataka.*

Nad šemom baze podataka **company**, koja je data sledećom slikom, kreirati veb aplikaciju prateći sledeće korake:

1. Kreirati veb projekat pod nazivom JPA-vezba7;
2. Kreirati **MySQL** bazu podataka pod nazivom company;
3. Izvršiti u kreiranom projektu povezivanje na datu bazu, a zatim izvršiti fajl sa priloženim SQL upitima za kreiranje baze podataka;
4. Nad tabelama baze podataka kreirati **JPA** entitete;
5. Nad JPA entitetima kreirati kontrolere;
6. Nad JPA entitetima kreirati **JSF** stranice;
7. Pokrenuti aplikaciju.

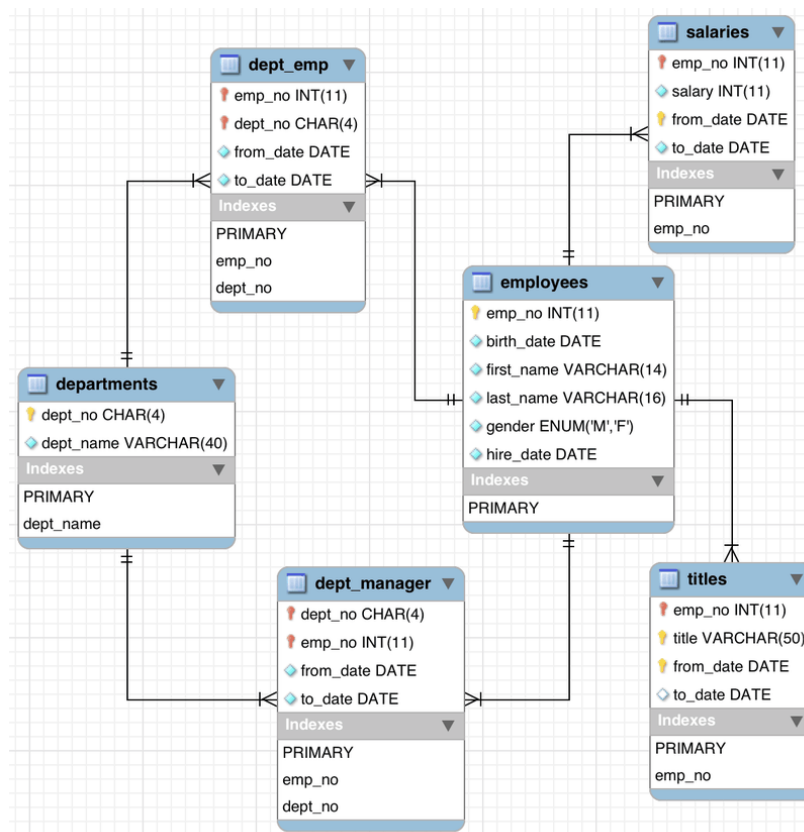
```
CREATE TABLE employees (  
    emp_no      INT          NOT NULL,  
    birth_date  DATE          NOT NULL,  
    first_name  VARCHAR(14)   NOT NULL,  
    last_name   VARCHAR(16)   NOT NULL,  
    gender      ENUM ('M','F') NOT NULL,  
    hire_date   DATE          NOT NULL,  
    PRIMARY KEY (emp_no)  
);  
  
CREATE TABLE departments (  
    dept_no     CHAR(4)       NOT NULL,  
    dept_name   VARCHAR(40)   NOT NULL,  
    PRIMARY KEY (dept_no),  
    UNIQUE KEY (dept_name)  
);  
  
CREATE TABLE dept_manager (  
    dept_no     CHAR(4)       NOT NULL,  
    emp_no      INT          NOT NULL,  
    from_date   DATE          NOT NULL,  
    to_date     DATE          NOT NULL,  
    KEY         (emp_no),  
    KEY         (dept_no),  
    FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,  
    FOREIGN KEY (dept_no) REFERENCES departments (dept_no) ON DELETE CASCADE,
```

```
        PRIMARY KEY (emp_no,dept_no)
    );

CREATE TABLE dept_emp (
    emp_no      INT           NOT NULL,
    dept_no     CHAR(4)       NOT NULL,
    from_date   DATE          NOT NULL,
    to_date     DATE          NOT NULL,
    KEY         (emp_no),
    KEY         (dept_no),
    FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
    FOREIGN KEY (dept_no) REFERENCES departments (dept_no) ON DELETE CASCADE,
    PRIMARY KEY (emp_no,dept_no)
);

CREATE TABLE titles (
    emp_no      INT           NOT NULL,
    title       VARCHAR(50)   NOT NULL,
    from_date   DATE          NOT NULL,
    to_date     DATE,
    KEY         (emp_no),
    FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
    PRIMARY KEY (emp_no,title, from_date)
);

CREATE TABLE salaries (
    emp_no      INT           NOT NULL,
    salary      INT           NOT NULL,
    from_date   DATE          NOT NULL,
    to_date     DATE          NOT NULL,
    KEY         (emp_no),
    FOREIGN KEY (emp_no) REFERENCES employees (emp_no) ON DELETE CASCADE,
    PRIMARY KEY (emp_no, from_date)
);
```

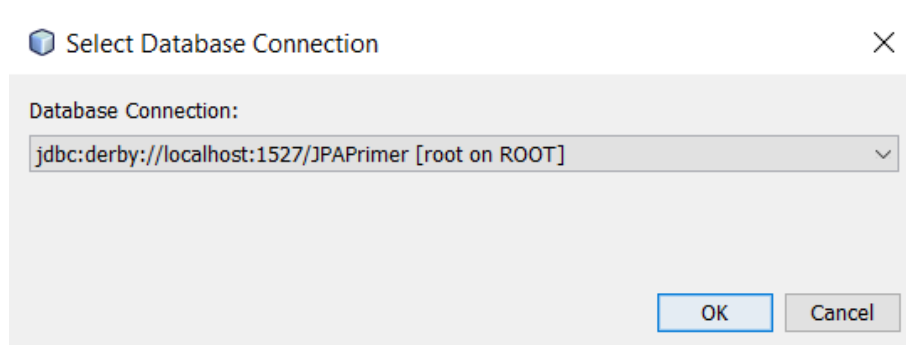


Slika 9.1 Šema baze podataka company [izvor: autor]

## IZVRŠAVANJE UPITA I KREIRANJE TABELA BAZE PODATAKA

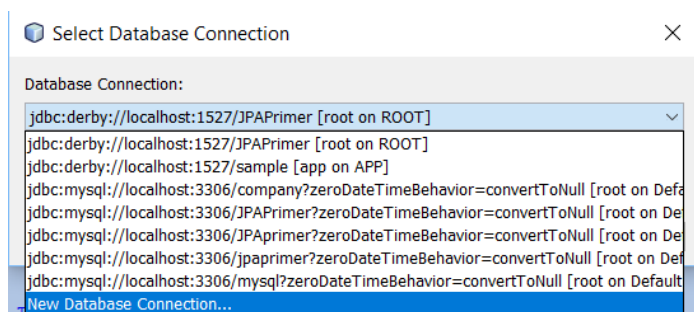
*U razvojnom okruženju je neophodno izvršiti datoteku sa SQL upitima.*

U projektu je kreiran folder SQL i u njega je ubačen fajl koji je dobio naziv `create_populate_tables.sql` i koji sadrži kod sa prethodne sekcije. Klikom na dugme Run SQL otvara se prozor za izbor DB konekcije:



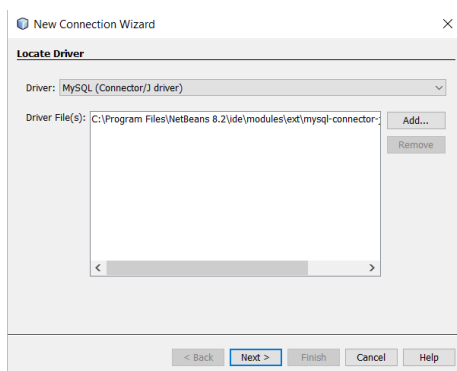
Slika 9.2 Izbor DB konekcije [izvor: autor]

U padajućem meniju sa slike, neophodno je izvršiti povezivanje na bazu podataka `company`.



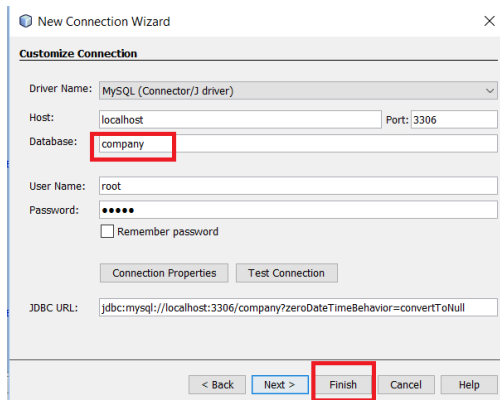
Slika 9.3 New DB konekcija [izvor: autor]

Bira se MySQL drajver.



Slika 9.4 Izbor tipa baze podataka [izvor: autor]

Unosi se naziv baze i izvršava se lista upita.



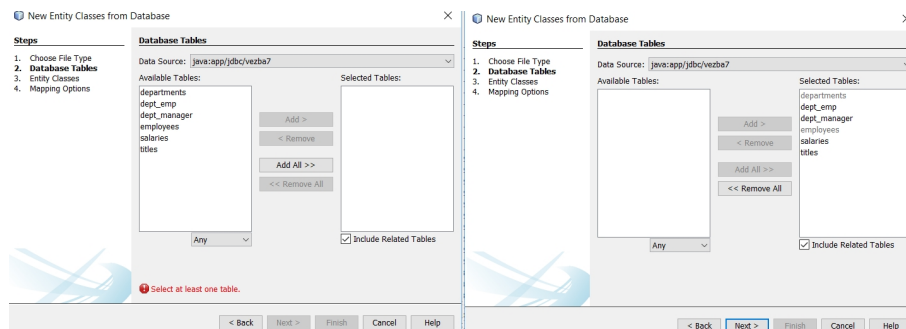
Slika 9.5 Povezivanje na željenu bazu podataka. [izvor: autor]

## KREIRANJE JPA ENTITETA

*Sledi kreiranje JPA entiteta nad tabelama baze podataka.*

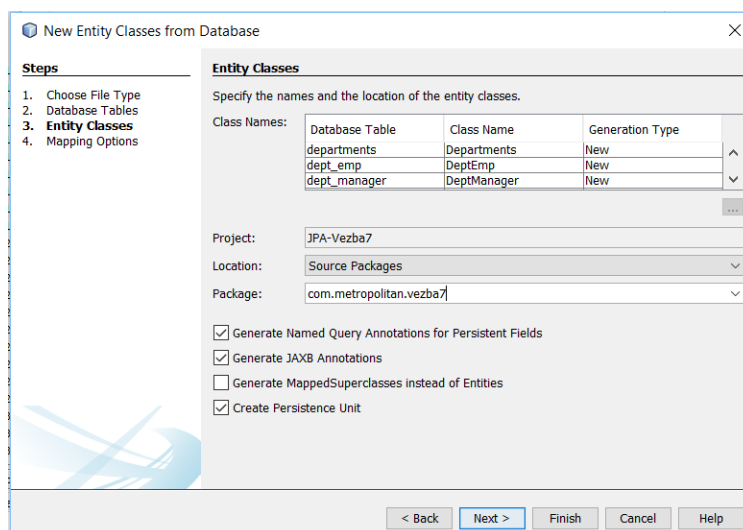
Kreirane su tabele baze podataka. U novom projektu će biti izabrana opcija **File | New**, a zatim se kreira datoteka iz kategorije **Persistence** i tipa **Entity Classes from Database** (o svemu ovome je već bilo govora).

U prozoru **New Entity Classes from Database** biraju se tabele od kojih je neophodno kreirati JPA entitete (sledeća slika).



Slika 9.6 Izbor tabela za kreiranje JPA entiteta [izvor: autor]

Nakon izabranih tabela klikom na **Next** otvara se prozor u kojem se navodi naziv paketa kojem će kreirane klase pripadati.

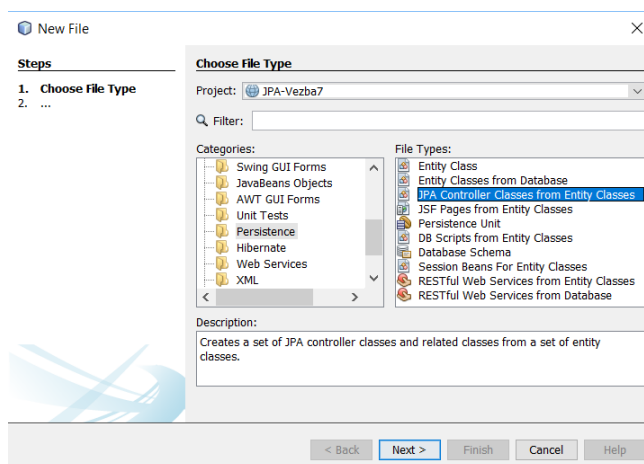


Slika 9.7 Definisane paketa i kraj kreiranja JPA klasa [izvor: autor]

## KREIRANJE KONTOLERA JPA ENTITETA

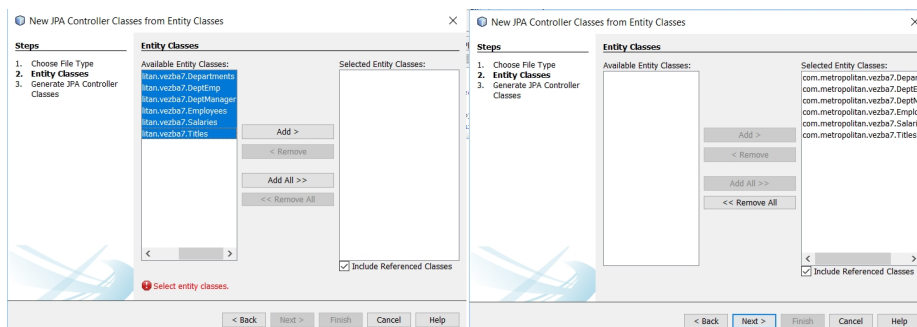
*Dao klase za upravljanje podacima su sledeće koje se kreiraju.*

Po dobro poznatom scenariju, koristeći razvojno okruženje, bira se opcija **File | New** i otvara se poznati prozor za izbor kategorije i tipa aplikacije. U ovom slučaju kao kategorija se bira **Persistence category**, dok će **JPA Controller Classes from Entity Classes** predstavljati tip datoteke koja se kreira.



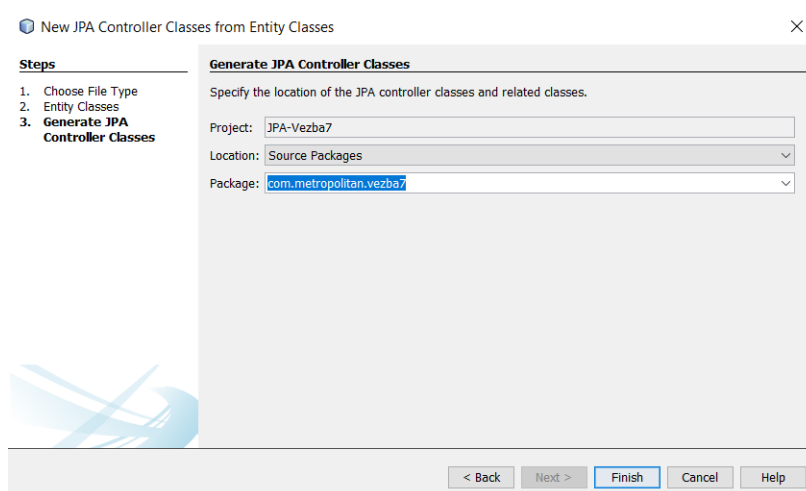
Slika 9.8 Izbor za kreiranje JPA kontroler klasa [izvor: autor]

Dalje se otvara prozor za dodavanje JPA klasa entiteta za koje se kreiraju kontroleri.



Slika 9.9 Izbor JPA klasa entiteta [izvor: autor]

Na kraju je neophodno izabrati paket kojem će pripadati kreirane klase.



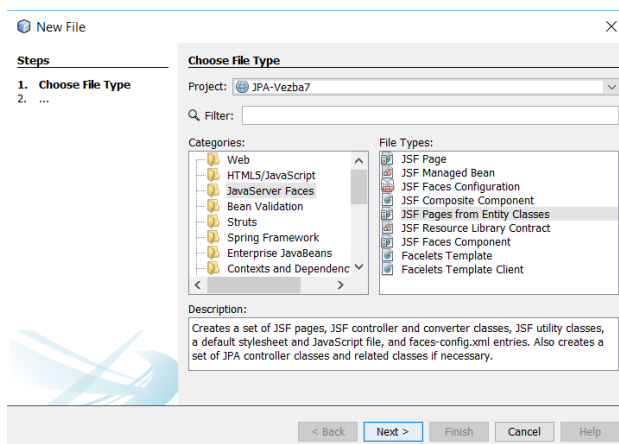
Slika 9.10 Izbor paketa za klase kontrolera [izvor: autor]

## KREIRANJE JSF STRANICA IZ JPA ENTITETA

*Veb aplikacija se zaokružuje generisanjem JSF stranica.*

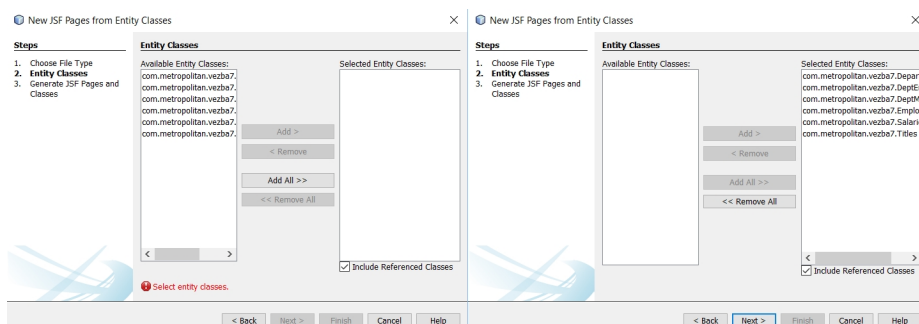


U prozoru New File bira se kategorija datoteke (u ovom slučaju JavaServer Faces ili Persistence) i tip datoteke (u ovom slučaju JSF Pages from Entity Classes) kao na sledećoj slici.



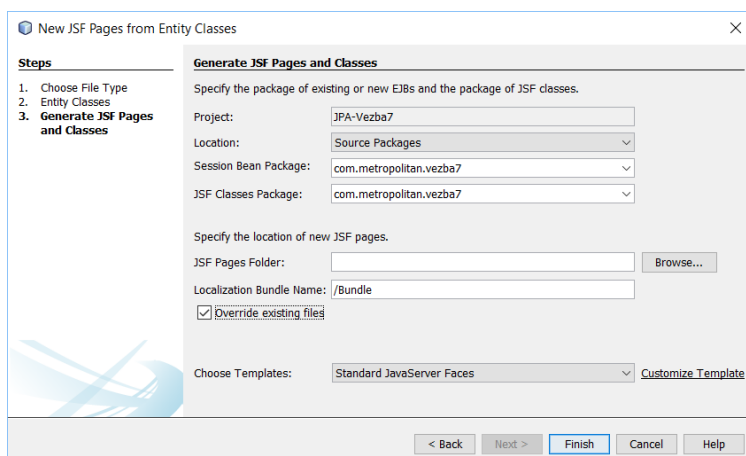
Slika 9.11 Izbor kreiranja JSF stranica iz JPA entiteta [izvor: autor]

Dalje, ponove se srećemo za prozorom za izbor JPA klasa entiteta, ali ovaj put za generisanje JSF stranica.



Slika 9.12 Izbor klasa JPA entiteta [izvor: autor]

Konačno biraju se paketi za zrna sesije i JSF klase. Aplikacija je generisana.



Slika 9.13 Kraj zadatka [izvor: autor]

## ✓ Poglavlje 10

### Individualna vežba 7

#### INDIVIDUALNA VEŽBA (135 MIN)

*Pokušajte sami*

##### **Zadatak 1 (45 min):**

1. Pokušajte da sredite GUI kreiranih JSF stranica iz prethodnog primera.
2. Pokušajte da iskoristite neke od obrađenih biblioteka komponenata.
3. Uposlite aplikaciju i proverite korektnost urađenih zadataka.

##### **Zadatak 2 (90 min):**

1. Uradite prethodni zadatak direktnim inženjeringom;
2. Kreirajte entitete;
3. Kreirajte jedinicu perzistencije;
4. Kreirajte odgovarajuće tabele baze podataka;
5. Kreirajte JSF poglede;
6. Pokušajte da iskoristite neke od obrađenih biblioteka komponenata.
7. Uposlite aplikaciju i proverite korektnost urađenih zadataka.

## ✓ Poglavlje 11

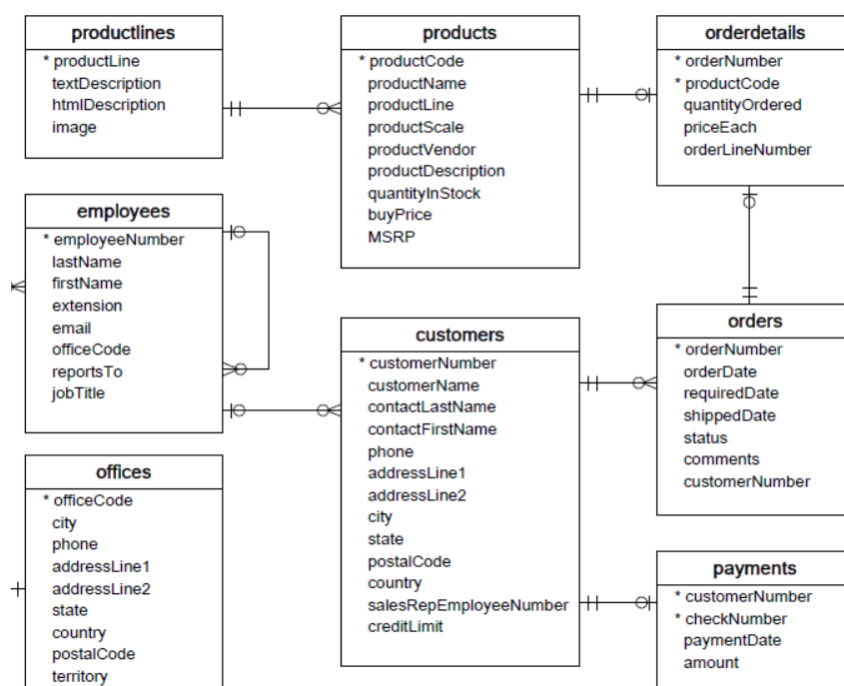
### Domaći zadatak 7

#### ZADATAK 1 (180 MIN)

##### *Izrada domaćeg zadatka.*

Po uzoru na zadatak sa vežbi kreirajte veb aplikaciju nad bazom podataka koja je data šemom sa sledeće slike, primenom:

1. obrnutog JPA inženjeringa;
2. direktnog JPA inženjeringa.



Slika 11.1 Šema baze podataka [izvor: autor]

Pored obaveznog DZ studenti dobijaju različite zadatke na mail od predmetnog asistenta.

## ▼ Poglavlje 12

# Zaključak

## ZAKLJUČAK

*Lekcija će se bavila radom sa bazama podataka primenom Java Persistence API (JPA).*

U prethodnom izlaganju lekcija je istakla da **Java Persistence API (JPA)** predstavlja API za objektno - relaciono mapiranje (**object - relational mapping** - ORM). Takođe, posebno je u lekciji naglašeno da ORM alati pomažu prilikom automatizovanja mapiranja Java objekata u tabele relacionih baza podataka. Prve verzije J2EE koristile su entitetska zrna kao standardni ORM pristup. Entitetska zrna (entity beans) su uvek pokušavala da drže sinhronizovanim podatke koji se čuvaju u memoriji sa podacima baza podataka. Iako je ovo odlična ideja, u praksi je pokazala ozbiljne nedostatke koji su se reflektovali kroz loše performanse aplikacija.

Lekcija ističe da je JPA integrisala ideje nekoliko ORM alata i postavila ih kao standard. Upravo tim alatima se ova lekcija detaljno bavila.

Lekcija je stavila fokus na sledeće teme:

- Kreiranje JPA entiteta;
- Interakcija sa JPA entitetima primenom klase EntityManager;
- Generisanje JPA entiteta iz postojećih šema baza podataka;
- Korišćene JPA upita i Java Persistence Query Language (JPQL);
- Kreiranje JSF aplikacije sa JPA entitetima.

Savladavanjem ove lekcije studenti razumeju ORM pristup i vladaju primenom ORM alata u radu sa bazama podataka složenih Java EE veb aplikacija..

## LITERATURA

*Za pripremu lekcije korišćena je najnovija literatura.*

1. Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. 2014. Java Platform, Enterprise Edition The Java EE Tutorial, Release 7, ORACLE
2. David R. Heffelfinger. 2015. Java EE7 Development With NetBeans 8, PACK Publishing
3. Yakov Fain. 2015. Java 8 programiranje, Kombib (Wiley)
4. Josh J. Ueneau. 2015. Java EE7 Recipes, Apress
5. <https://www.tutorialspoint.com/jsf/>
6. <https://www.tutorialspoint.com/jpa/>

7. <http://www.mysqltutorial.org/mysql-sample-database.aspx>