

3. Laborator JPA

3.1 Apache Derby DB

Derby DB este o baza de date open-source, implementata integral in java de catre fundatia Apache (<https://db.apache.org/derby/>)

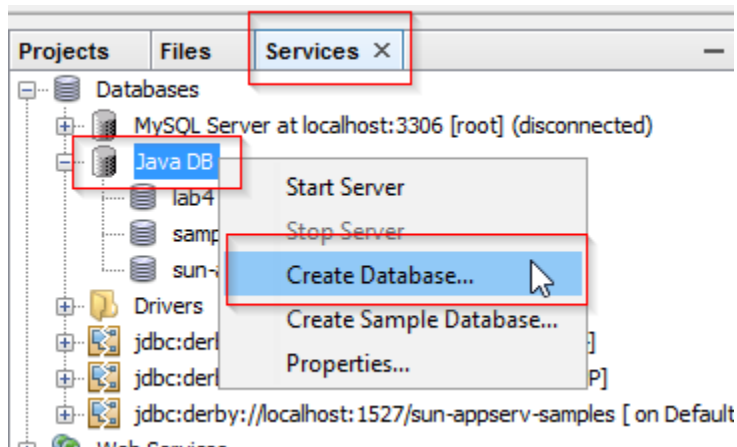
Are un foot-print mic (3.5 MB) in care este inclus motorul de baza de date si driverul embedded JDBC.

Datorita acestui driver Derby poate fi 'embeddata' in orice aplicatie java.

Derby este inclusa in distributia NetBeans astfel incat nu este nevoie sa instalam suplimentar nimic.

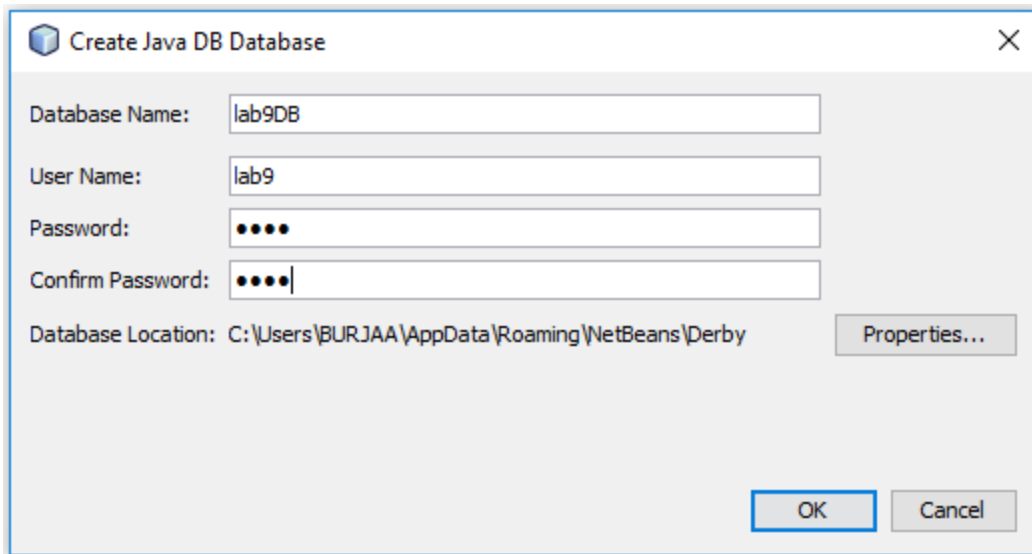
3.1.1 Crearea unei baze de date

Pentru a crea o noua baza de date selectati tab-ul 'Services' (stanga sus):



Creati baza de date cu numele **lab3DB**, user: **lab3**, pass: **lab3** :

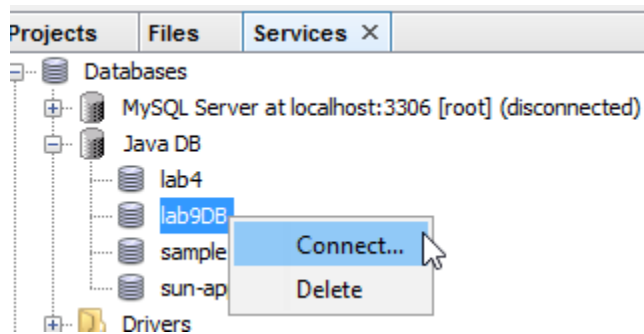
Lab 3



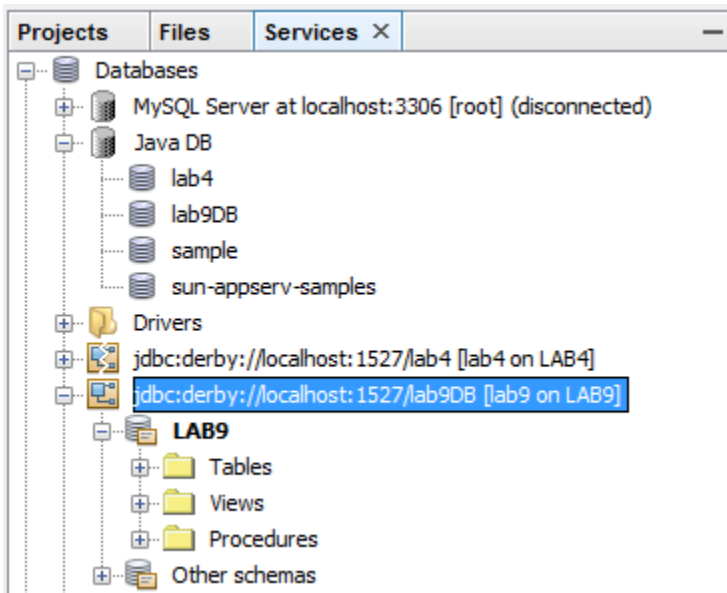
The 'Create Java DB Database' dialog box is shown. It contains the following fields and buttons:

- Database Name: lab9DB
- User Name: lab9
- Password: [masked with dots]
- Confirm Password: [masked with dots]
- Database Location: C:\Users\BURJAA\AppData\Roaming\NetBeans\Derby
- Buttons: Properties..., OK, Cancel

Apoi porniti aceasta baza de date:

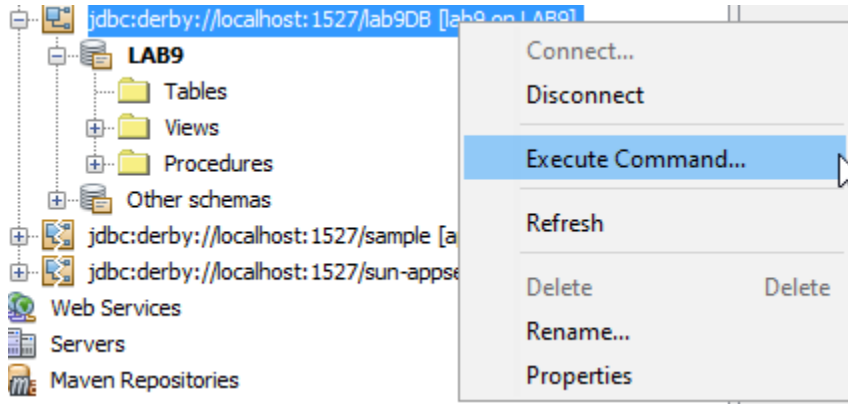


Dupa acest pas avem disponibila conexiunea la baza de date:



Lab 3

Iar acum putem executa comenzi SQL:



3.1.2 Operatii CRUD

CRUD = Create Read Update Delete

Creati tabela "Department" si executati fiecare din operatiile CRUD de mai jos:

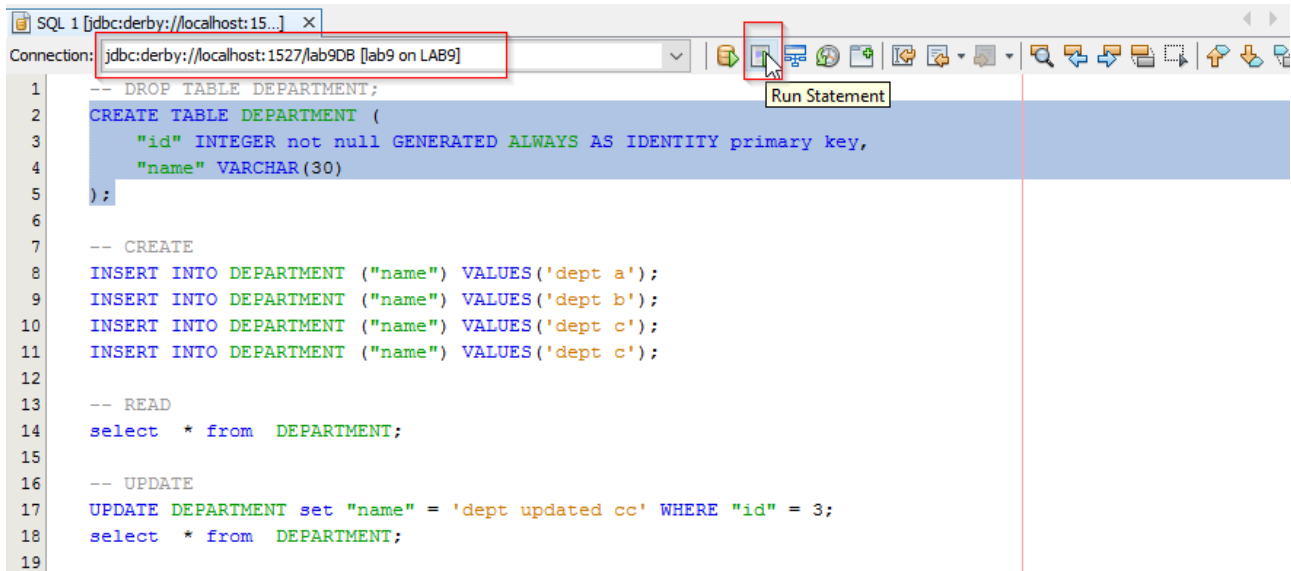
```
-- DROP TABLE DEPARTMENT;  
CREATE TABLE DEPARTMENT (  
    "ID" INTEGER not null GENERATED ALWAYS AS IDENTITY primary key,  
    "NAME" VARCHAR(30)  
);  
  
-- CREATE  
INSERT INTO DEPARTMENT ("NAME") VALUES('dept a');  
INSERT INTO DEPARTMENT ("NAME") VALUES('dept b');  
INSERT INTO DEPARTMENT ("NAME") VALUES('dept c');  
INSERT INTO DEPARTMENT ("NAME") VALUES('dept c');  
  
-- READ  
select * from DEPARTMENT;  
  
-- UPDATE  
UPDATE DEPARTMENT set "NAME" = 'dept updated cc' WHERE "id" = 3;  
select * from DEPARTMENT;  
  
-- DELETE  
DELETE from DEPARTMENT where "NAME" = 'dept a';  
select * from DEPARTMENT;  
  
-- SEARCH  
select * from DEPARTMENT where "NAME" = 'dept c';  
  
-- FILTER  
select * from DEPARTMENT where "NAME" like '%update%';
```

In scop didactic executati pe rand fiecare comanda .

Pentru executia doar a unei comenzi sau a unui grup de comenzi:

Lab 3

- selectati comanda/comenzile
- apasati butonul „Run statement”



```
SQL 1 [jdbc:derby://localhost:15...] X
Connection: jdbc:derby://localhost:1527/lab9DB [lab9 on LAB9]

1  -- DROP TABLE DEPARTMENT;
2  CREATE TABLE DEPARTMENT (
3      "id" INTEGER not null GENERATED ALWAYS AS IDENTITY primary key,
4      "name" VARCHAR (30)
5  );
6
7  -- CREATE
8  INSERT INTO DEPARTMENT ("name") VALUES ('dept a');
9  INSERT INTO DEPARTMENT ("name") VALUES ('dept b');
10 INSERT INTO DEPARTMENT ("name") VALUES ('dept c');
11 INSERT INTO DEPARTMENT ("name") VALUES ('dept c');
12
13 -- READ
14 select * from DEPARTMENT;
15
16 -- UPDATE
17 UPDATE DEPARTMENT set "name" = 'dept updated cc' WHERE "id" = 3;
18 select * from DEPARTMENT;
19
```

Urmarii pas cu pas continutul tabelii cu ajutorul comenzii READ (select * from tbl_name);

Nota:

1. pentru Derby, numele coloanelor se marcheaza cu ghilimele (ex. "name") iar continutul cu apostroafe (ex. 'dept c').
2. Se recomanda ca fiecare tabela sa aiba un camp Id de tip integer/long automatic generat si auto-increment. Acesta va fi completat de baza de date automat si incrementat la fiecare insertie. Nu se va decrementa la stergerea intrarilor.

3.1.2 Tabele relationate (in cadrul bazelor de date relationale)

In continuare vom create o noua tabela numita „Employee” si care va relationa tabela „Department”:

```
-- DROP TABLE EMPLOYEE;
CREATE TABLE EMPLOYEE (
    "ID" INTEGER not null GENERATED ALWAYS AS IDENTITY primary key,
    "FIRSTNAME" VARCHAR(30),
    "LASTNAME" VARCHAR(30),
    "IDDEPARTMENT" INTEGER CONSTRAINT dept_fk REFERENCES DEPARTMENT ("ID") ON
    DELETE SET NULL
);

-- CREATE
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES ('user 1',
    'popescu', 3);
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES ('user 2',
    'popa', 3);
```

Lab 3

```
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES ('user 3',
    'moraru', 4);
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES ('user 4',
    'muntean', 2);
INSERT INTO EMPLOYEE ("FIRSTNAME", "LASTNAME", "IDDEPARTMENT") VALUES ('user 5',
    'besoiu', 2);

-- READ
SELECT * FROM EMPLOYEE;

SELECT emp.*, dept."NAME"
    FROM EMPLOYEE AS emp
    LEFT JOIN DEPARTMENT as dept on emp."IDDEPARTMENT" = dept."ID"
;

-- cautare/filtrare
SELECT emp.*, dept."name"
    FROM EMPLOYEE AS emp
    LEFT JOIN DEPARTMENT as dept on emp."IDDEPARTMENT" = dept."ID"
WHERE emp."LASTNAME" like '%O%'
;

-- UPDATE
UPDATE EMPLOYEE set "FIRSTNAME" = 'user update new' WHERE "LASTNAME" = 'moraru';

-- DELETE
DELETE FROM EMPLOYEE WHERE "FIRSTNAME" = 'user 2' OR "LASTNAME"='popescu';
```

3.2 JPA, Entitati JPA

JPA = Java Persistence API. Set de metode (API) care faciliteaza transferul datelor catre/dinspre baza de date.

Obiectele 'Entity' sunt instante in memorie a claselor entitate si care contin datele efective din baza de date. In JPA obiectele entitate se folosesc pentru operatii CRUD (create, read, update, delete).

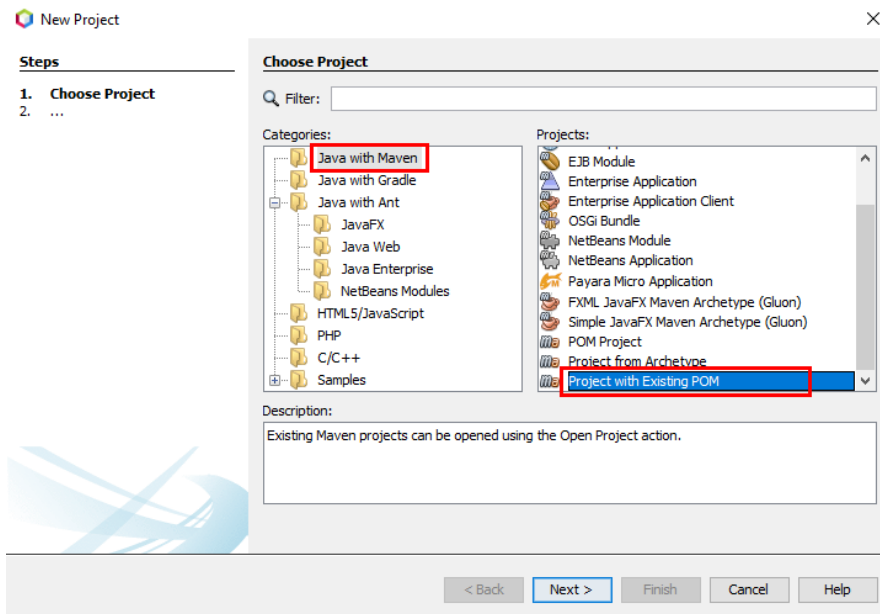
Pentru aceste operatiuni sunt necesare deci 3 elemente:

- Entitati (clase)
- Persistence.xml
- Clase (clienti) care opereaza cu cu aceste entitati

3.3 Aplicatia

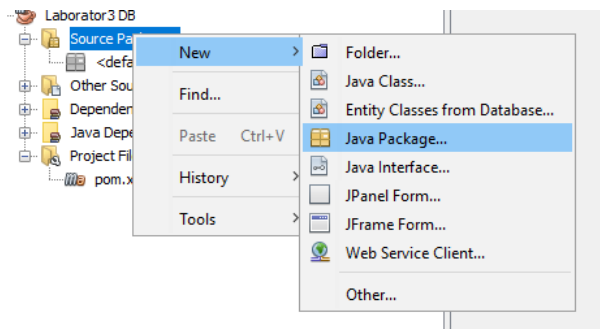
Dezarhivati arhiva lab3_starter.zip pe disc, si apoi din Netbeans deschideti acest proiect maven, cu optiunea 'Project with existing POM' / 'existing sources'.

Lab 3



3.3.1 Adaugare entitati

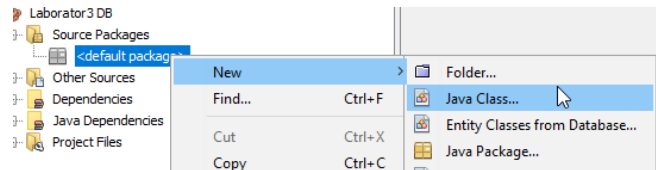
Creati un nou pachet unde vom stoca sursele: right-click > New > Other > Java Package



Numele pachetului: ro.ulbs.ism.an3

Creati 2 clase (entitati): Department si Employee pentru structura tabelor de mai de sus.

Selectati pachetul creat > right click > New > Java Class



Si creati sursele:

```
@Entity
@Table(name = "DEPARTMENT")
public class Department implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
```

Lab 3

```
@GeneratedValue(strategy = GenerationType.IDENTITY)
@Column(name = "id")
private Integer id;

@Column(name = "name")
private String name;
}

@Entity
@Table(name = "EMPLOYEE")
public class Employee implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "id")
    private Integer id;

    @Column(name = "firstname")
    private String firstname;

    @Column(name = "lastname")
    private String lastname;

    @JoinColumn(name = "iddepartment", referencedColumnName = "id")
    @ManyToOne
    private Department department;
}
```

Fiecarei clase generati constructor, getters si setters:

Va pozitionati cu cursorul dupa ultimul camp declarat, apoi ALT+INSERT > Constructor. Nu selectati nimic.

Va pozitionati cu cursorul dupa constructorul generat, apoi ALT+ INSERT > Getter and Setter Selectati toate campurile

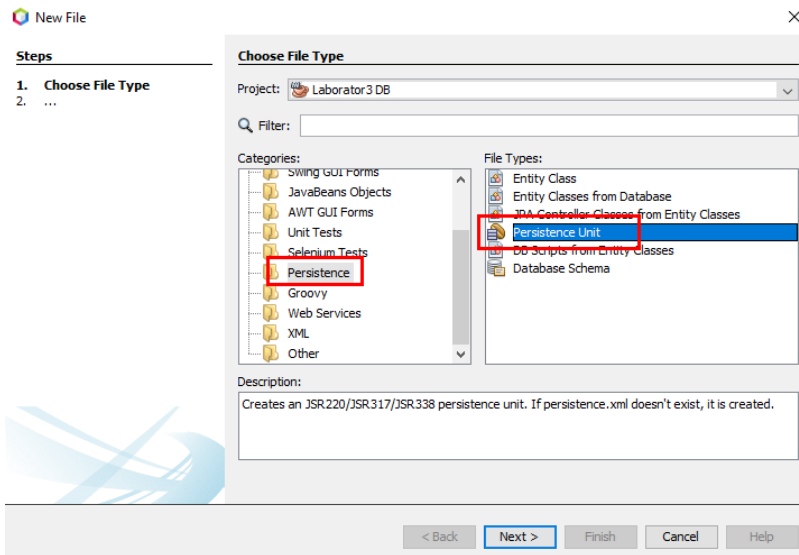
Tot cu ALT+INSERT veti putea adauga importurile necesare.

ALT+INSERT = Source > Insert code ...

3.3.2 Adaugare persistence unit

Right-click proiect > New > Other > Persistence > Persistence Unit

Lab 3

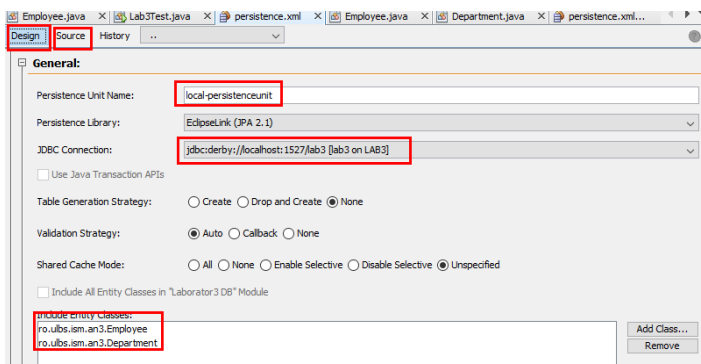
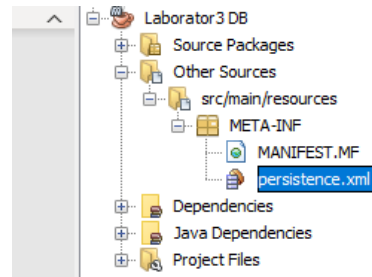
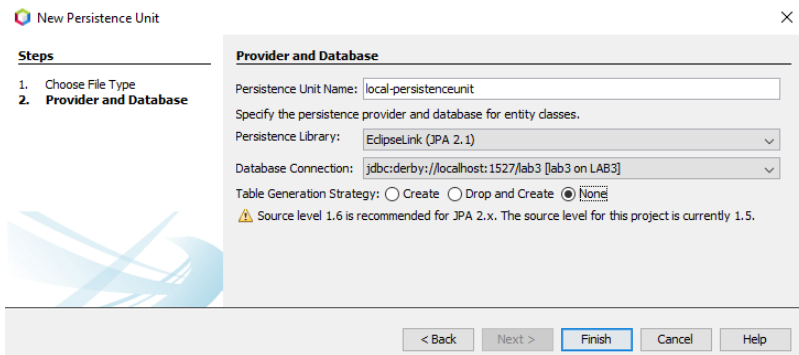


Nume: local-persistenceunit; EclipseLink; no table generation strategy.
Rezultatul este fisierul src/main/resources/META-INF/persistence.xml

In acest fisier adaugam lista claselor de tip Entity:

```
<class>ro.ulbs.ism.an3.Employee</class>
```

```
<class>ro. ulbs.ism.an3.Department</class>
```



Lab 3

Valori aditionale de adaugat in sursa persistence.xml:

```
<property name="eclipselink.target-database" value="Derby"/>
<property name="eclipselink.platform.class.name" value="org.eclipse.persistence.platform.database.DerbyPlatform"/>
<property name="eclipselink.weaving" value="static"/>
<property name="eclipselink.jpa.uppercase-column-names" value="true"/>
<!-- Optional enable debugging sql queries and parameters -->
<!-- <property name="eclipselink.logging.level.sql" value="FINE"/> -->
<!-- <property name="eclipselink.logging.parameters" value="true"/> -->
```

3.3.3 Adaugare client/clasa test

Generam o noua clasa in pachetul ro.ulbs.ism.an3, numita Laborator3Test in care definim metoda main:

```
public static void main(String[] args) {
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("local-persistenceunit");
    EntityManager em = emf.createEntityManager();

    em.getTransaction().begin();

    Department depA = new Department();
    depA.setName("Departament A");
    em.persist(depA);

    Department depB = new Department();
    depB.setName("Departament B");
    em.persist(depB);

    em.getTransaction().commit();

    em.close();
    emf.close();
}
```

Remarcati faptul folosim denumirea unitului de persistenta definit in persistence.xml: "local-persistenceunit"

Similar se poate adauga cod pentru a crea instante Employee:

```
Employee e1 = new Employee();
e1.setFirstname("Ursu");
e1.setLastname("Nicola");
em.persist(e1);

Employee e2 = new Employee();
e2.setFirstname("Ion");
e2.setLastname("Oarga");
e2.setDepartment(depA);
em.persist(e2);

Employee e3 = new Employee();
e3.setFirstname("Marcu");
e3.setLastname("Giurgiu");
e3.setDepartment(depB);
em.persist(e3);
```

Pentru a afisa datele existente in baza de date, dupa commit trebuie adaugat urmatorul cod:

```
System.out.println("===== AFISARE: =====");
Query query = em.createQuery("SELECT o FROM Employee o");
List<Employee> employees = query.getResultList();
String deptDetails;
for(Employee e: employees) {
    deptDetails = "";
    if (e.getDepartment() != null) {
        deptDetails = " departament: "+e.getDepartment().getName();
    }
}
```

Lab 3

```
System.out.println("E: [" + e.getFirstname() + " " + e.getLastname() + " " + deptDetails + ""]");  
}
```

Sumar

Entitatile JPA sunt clase care au particularitatile:

La nivelul intregii clase :

Serializabile (implementeaza java.io.Serializable)

@Entity

@Table(name = "...")

Optional la nivelul fiecarui camp:

@Column(name = "id")

@Id

@OneToMany

@ManyToOne

@JoinColumn

Elementul de persistenta este definit in persistence.xml cu nume si tip. Tipul detaliilor de persistenta pentru exemplul de aici este *RESOURCE_LOCAL*.

Tot in persistence.xml trebuie sa definim lista claselor de tip Entity.

Persistence.xml determina modul in care definim detaliile la sistemul de persistenta (db), lista elementelor persistente (entity), parametrii ce afecteaza modul de persistare a datelor

Problema :

Extindeti metoda main() pentru a efectua si operatii update (em.merge()) si stergere (em.delete()) asupra unor entitati.

Ex.

- cautare entitate Employee cu id-ul 3, schimbare nume, actualizare si apoi afisarea tuturor entitatilor Employee din DB

- cautare entitate Employee cu id-ul 4, stergere, si apoi afisarea tuturor entitatilor Employee din DB