Loeng 9

JPA

Analoogia

- Entity Framework

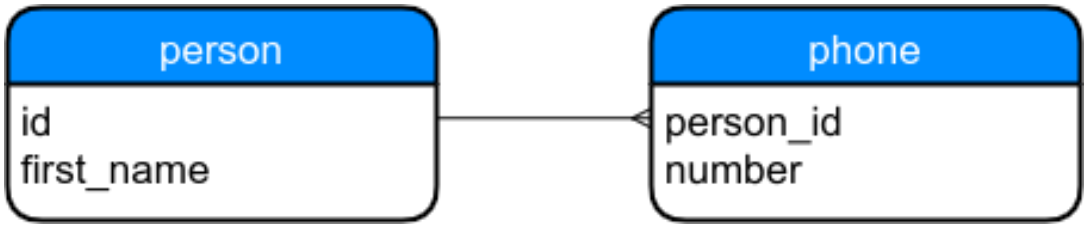
Java Persistance API (JPA)

- Java ORM-i standard

Object Relational Mapping (ORM)

```
public class Person {
    private Long id;
    private String name;
    private List<Phone> phones;
}
```

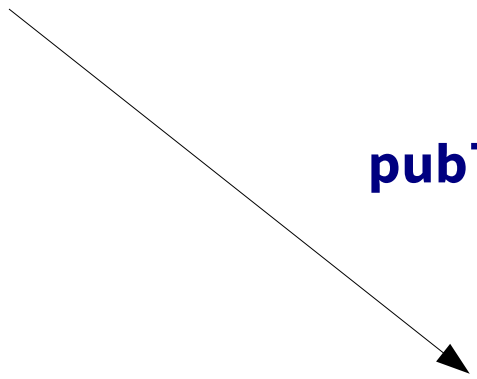
Object-relational gap



Miks?

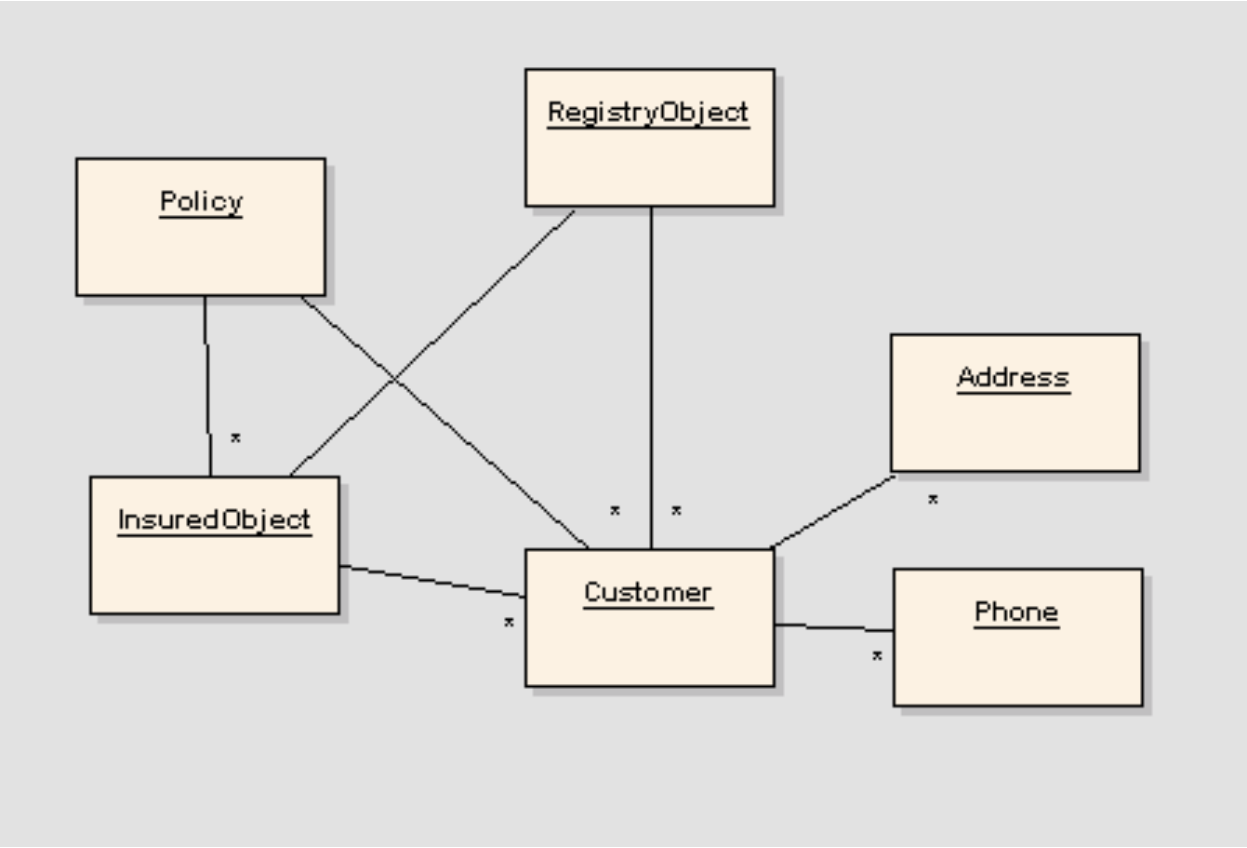
- Jdbc on kohmakas aga selle lahendab nt. Spring Data JDBC, QueryDsl jne.
- Probleem on baasikeskne lähenemine

```
public class Employee {
    private Long id;
    private String name;
    private Department department;
}
```



```
public class Employee {
    private Long id;
    private String name;
    private Long departmentId;
}
```

Keerulised objektide graafid

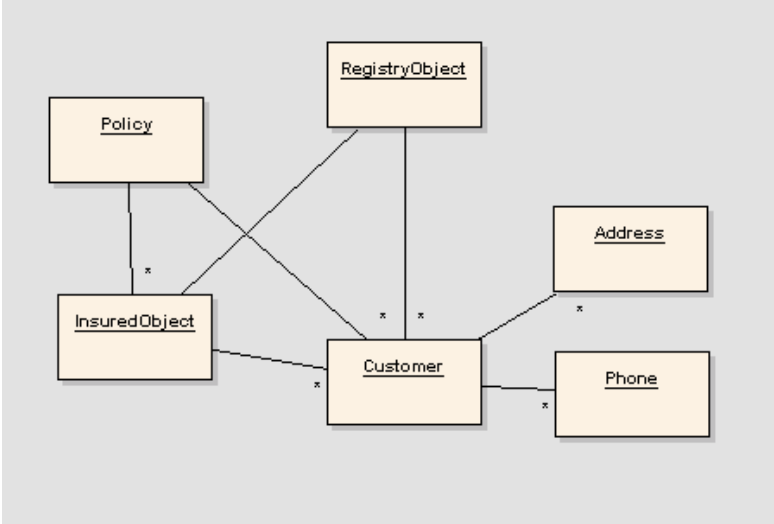


Keerulised objektide graafid

1. Lugeda poliis
2. Valideerida poliis
3. Arvutada poliisi hind

Baasikesksus

```
public Policy load(Long policyId);
public ValidationResult validatePolicy(Long policyId);
public Money calculatePremium(Long policyId);
```



- Efektiivsus (korduv töö)
- Disain (sõltuvus andmebaasist)
- Keerukus, andmebaas kui globaalne muutuja

Baasikesksus koodis

```
policy.setStartDate(date);
```

```
String sql = "UPDATE policy SET start_date = ? WHERE id = ?";

template.update(sql, date, id);
```

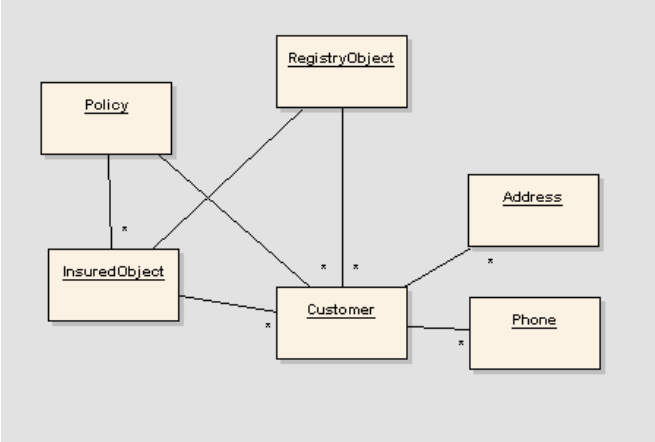
Baasikesksus koodis

```
policy.add(insuredObject);
policy.setCustomer(customer);
policy.setStartDate(date)
policy.calculateInstallments();
```

```
String sql = "...";

...
...
```

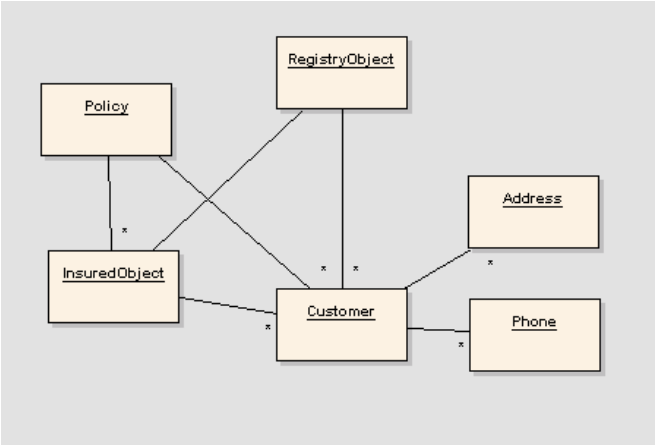
Ideaalis

```
Policy policy = db.load(policyId);

policy.add(insuredObject);
policy.setCustomer(customer);
policy.setStartDate(date)
policy.calculateInstallments();

validatePolicy(policy);

Money price = calculatePremium(policy);

// jne.

db.save(policy);
```



Lugemine

```
Policy policy = db.load(policyId);
```



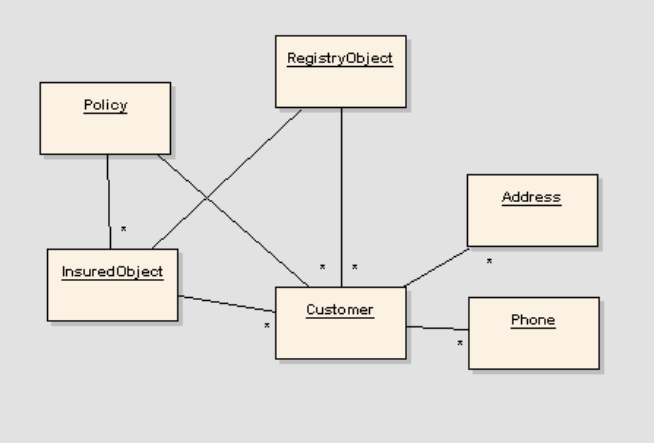
Lugemine

- Repository/DAO muster
- QueryDsl

```
JdbcUtil.getQueryFactory().update(qCustomer)
        .set(qCustomer.firstName, customer.getFirstName())
        .set(qCustomer.lastName, customer.getLastName())
        .set(qCustomer.code, customer.getCode())
        .where(qCustomer.id.eq(customer.getId()))
        .execute();
```

Salvestamine

```
policy.setCustomer(customer);
policy.setStartDate(date)
policy.calculateInstallments();

db.save(policy);
```



Salvestamine

- Objektorienteeritud andmebaasid
- NoSql andmebaasid
- ORM

ORM

- JPA (Hibernate, EclipseLink, OpenJPA, etc)

JPA

```
@Entity
public class Customer {

    @Id
    private Long id;

    private String firstName;

    private String lastName;

    @ElementCollection
    private List<Phone> phones;
```

JPA

```
Policy policy = jpa.load(policyId);

policy.add(insuredObject);
policy.setCustomer(customer);
policy.setStartDate(date)
policy.calculateInstallments();

validatePolicy(policy);

Money price = calculatePremium(policy);

// jne.

jpa.save(policy);
```

Olem (Entity)

```
import javax.persistence.*;

@Entity
public class Person {
    @Id
    private Long id;
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public Person() {} // ARGUMENTIDETA KONSTRUKTOR!!!

    public String getName() ...
    public void setName(String name) ...
}
```

JPA läbi Spring-i

```
implementation group: 'org.hibernate',
               name: 'hibernate-core',
               version: '5.4.22.Final'

implementation group: 'org.springframework',
               name: 'spring-orm',
               version: '5.2.9.RELEASE'
```

JPA läbi Spring-i

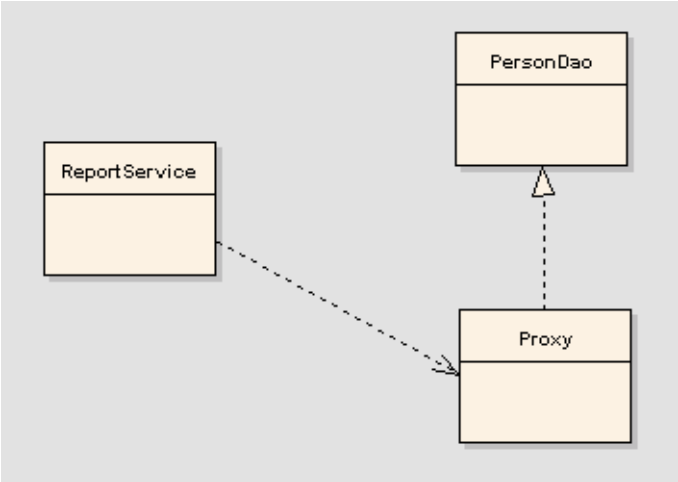
```
@PersistenceContext
private EntityManager em;

@Transactional
public void insert(Order order) {
    em.persist(order);
}
```

@Transactional

```
@Transactional
public void save(Order order) {

    ...

}
```

@Transactional

execution(public * ((@Transactional *)+).*(..));




```
@Repository
public class PersonDao {

    @PersistenceContext
    private EntityManager em;

    @Transactional
    public void insert(Person person) {
        em.persist(person);
    }
}
```

Olemi laadimine

```
em.find(Order.class, 1L);
```

Olemi muutmine

```
Order o = em.find(Order.class, 2L);

if (o != null) {
    o.setNumber("A123");
}
```

Olemi kustutamine

```
Employee employee = em.find(Employee.class, 1L);

if (employee != null) {
    em.remove(employee);
}
```

Päringud

```
Query query = em.createQuery("delete from Person");

query.executeUpdate();
```

Päringu parameetrid

```
TypedQuery<Employee> query = em.createQuery(
        "select e from Employee e where e.id = :id",
        Employee.class);

query.setParameter("id", 4L);

List<Employee> employees = query.getResultList();
```

JPQL

-- Java Persistence Query Language

NB! Ei ole SQL

- Räägib objektidest ja objekti väljadest
- Ei räägi baasi tabelistest ja väljadest
- Ei sõltu konkreetsest andmebaasist (nt. Oracle)

```
SELECT p FROM Player p WHERE p.name = :name
```

JPQL

```
SELECT p FROM Player p WHERE p.team.name like :team
```

JPQL

```
SELECT p.owner FROM Phone p WHERE p.number = :number
```

JPQL

```
SELECT e FROM Employee e WHERE e.subordinates IS empty
```

JPQL

https://docs.oracle.com/javaee/7/tutorial/persistence-querylanguage004.htm

Olemi salvestamine

```
Person jack = new Person("Jack");
em.persist(jack);
```

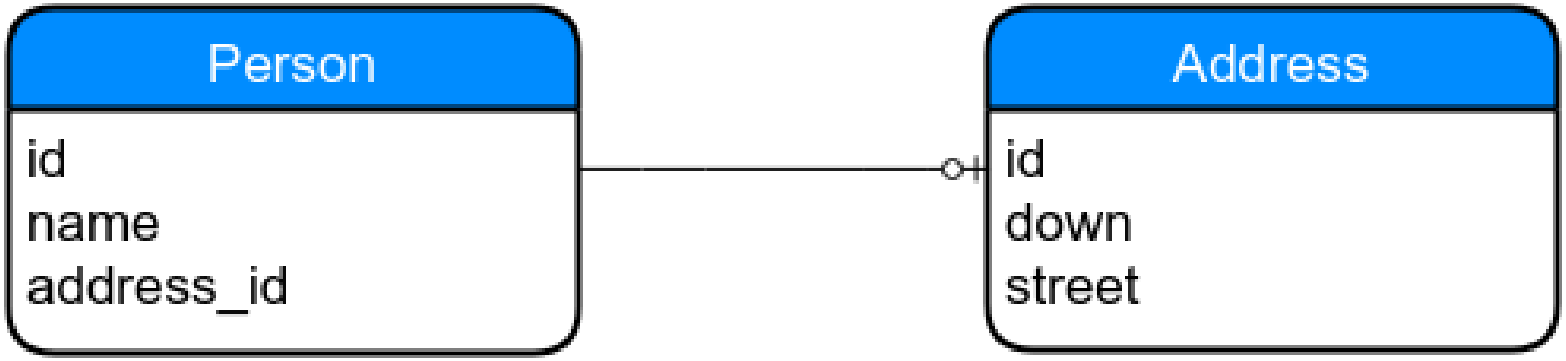
```
Person jill = new Person("Jill");
person.setId(1L);

em.persist(jill); // error
em.merge(jill); // ok
```

Olemi salvestamine

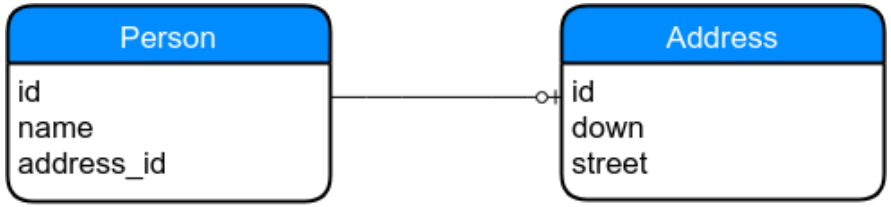
```
public void save(Person person) {

        if (person.getId() == null) {
            em.persist(person);
        } else {
            em.merge(person);
        }

    }
```

Peamised seosed

@OneToOne



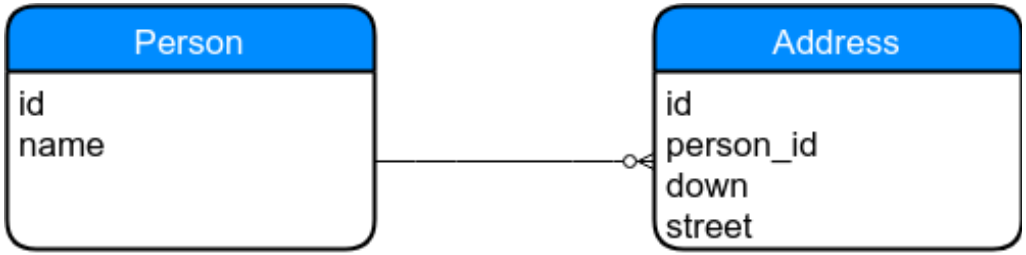
@OneToOne




```
@Entity
public class Person {
    @Id
    private Long id;

    private String name;

    @OneToOne
    private Address address;
```

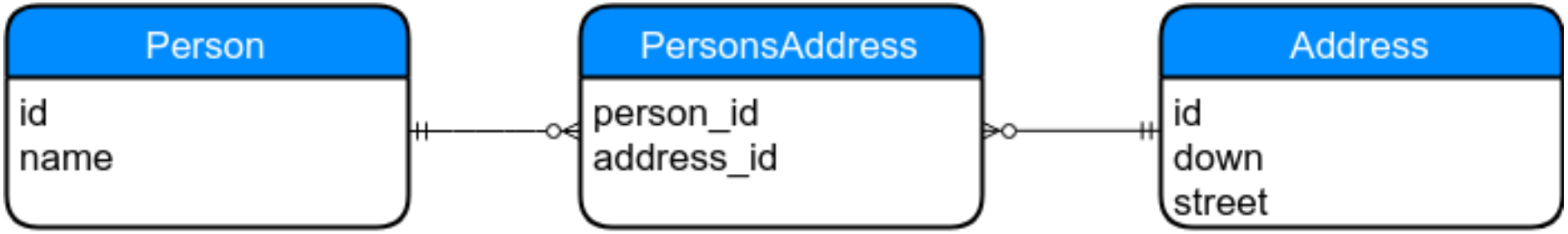
```
Person person = em.find(Person.class, 1L);

Address address = person.getAddress();
```

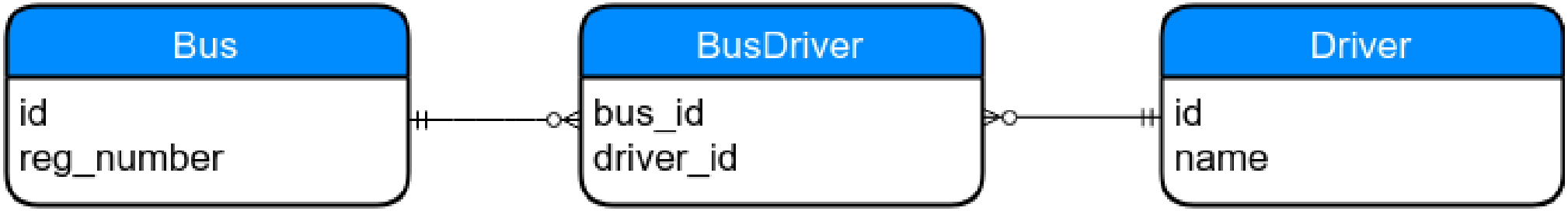
@OneToMany



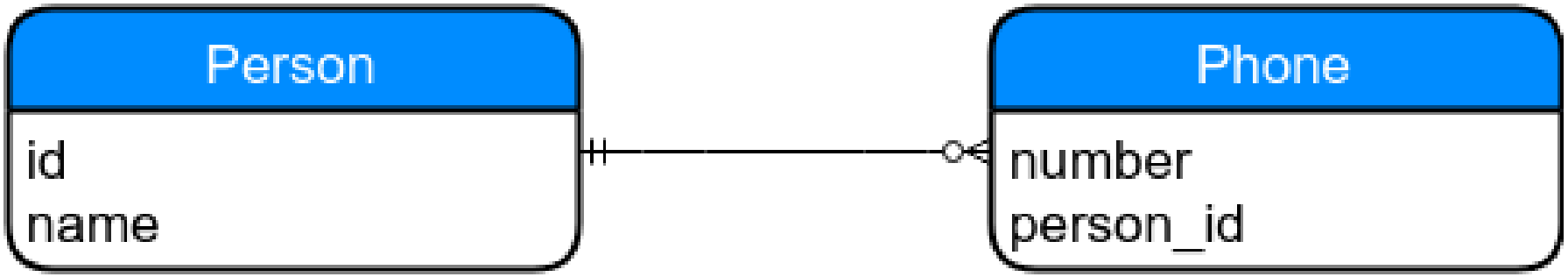

```
@Entity
public class Person {
    @Id
    private Long id;

    @OneToMany
    private List<Address> addresses;
```

@OneToMany, @ManyToMany

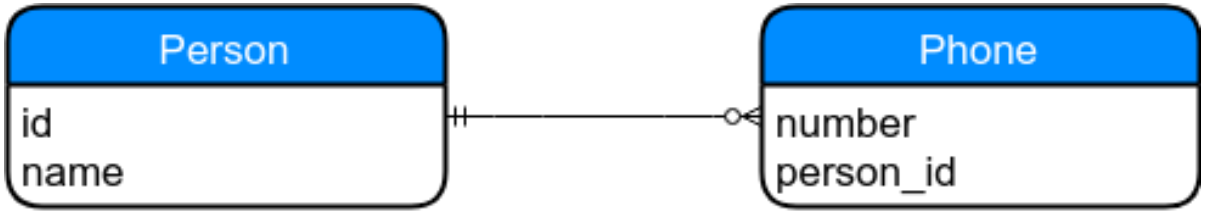




@ElementCollection



@ElementCollection




```
@Entity
public class Person {
    @Id
    private Long id;

    @ElementCollection
    private List<Phone> phones;
```

@ElementCollection

```
@Entity
public class Person {
    @Id
    private Long id;

    @ElementCollection
    private List<Phone> phones;
```

```
@Embeddable
public class Phone {
    // no id field
    private String number;
    ...
}
```

Id väli

```
@Entity
public class Employee {

    @Id
    @GeneratedValue
    private Long id;
```

Andmebaasi skeem

Alustamine baasi skeemist

- Hoiab rakendust koos
- Paremad võimalused optimeerimiseks
- Baasi skeem ja GUI mõjutavad rakenduse arhitektuuri

Alustamine koodist

- Mugavam
- Baasi skeem ei mõjuta rakenduse arhitektuuri ja hea koodi disaini tegemine on lihtsam

Kesktee

- Alustame koodist ning fikseerime ja optimeerime baasi skeemi hiljem
- Saame alustada parimast vaatest koodi poolel ja teha minimaalse vajaliku, et baasi pool piisavalt hea oleks.

Konfiguratsioon

```
@Bean
public EntityManagerFactory
            entityManagerFactory(DataSource dataSource) {

    LocalContainerEntityManagerFactoryBean factory =
            new LocalContainerEntityManagerFactoryBean();
    factory.setPersistenceProviderClass(
            HibernatePersistenceProvider.class);
    factory.setPackagesToScan("model");
    factory.setDataSource(dataSource);
    factory.setJpaProperties(additionalProperties());
    factory.afterPropertiesSet();

    return factory.getObject();
}
```

Konfiguratsioon

```
private Properties additionalProperties() {
    Properties properties = new Properties();

    properties.setProperty("hibernate.dialect",
      "org.hibernate.dialect.PostgreSQL10Dialect");

    properties.setProperty(
            "hibernate.show_sql", "true");

    properties.setProperty(
            "hibernate.hbm2ddl.auto", "create");


    return properties;
}
```

Skeemi loomine

```
properties.setProperty(
            "hibernate.hbm2ddl.auto", "create");
```

create - loob skeemi annotatsioonide järgi
validate – valideerib skeemi vastavust
update – muudab olemasolevat skeemi

Skeemi loomine

Projektis kasutame väärtust „validate”

```
properties.setProperty(
            "hibernate.hbm2ddl.auto", "validate");
```

Hsql Server

```
public static void main(String[] args) {
    Server server = new Server();

    server.setDatabasePath(0,
                           "mem:db1;sql.syntax_pgs=true");

    server.setDatabaseName(0, "db1");

    server.start();
}
```

Demo (kood vs baasi skeem)

Täpse skeemi määramine

```
@Entity
@Table(name = "isik")
public class Person {

    @Column(name = "nimi")
    private String name;
```

Täpse skeemi määramine

```
@Entity
public class Employee {

    @Id
    @GeneratedValue(strategy = GenerationType.SEQUENCE)
    private Long id;
```

Täpse skeemi määramine

```
@Id
@SequenceGenerator(name="my_seq",
                   sequenceName="SEQ1",
                   allocationSize=1)
@GeneratedValue(strategy = GenerationType.SEQUENCE,
                generator="my_seq")
private Long id;
```

Täpse skeemi määramine

```
@OneToOne
@JoinColumn(name = "aadressi_id")
private Address address;
```

Täpse skeemi määramine

```
@ElementCollection
@CollectionTable(
        name = "isiku_telefonid",
        joinColumns=@JoinColumn(name = "isiku_id",
                referencedColumnName = "id")
)
private List<Phone> phones = new ArrayList<>();
```

Cascade

```
@OneToOne(cascade = CascadeType.PERSIST)
private Address address;
```

```
Person person = new Person("Alice");
person.setAddress(new Address("Pine Street"));

em.persist(person);
```

Cascade

```
@OneToOne(cascade = { CascadeType.PERSIST,
                      CascadeType.MERGE,
                      CascadeType.REMOVE })
private Address address;
```

```
@OneToOne(cascade = CascadeType.ALL)
private Address address;
```

Fetch Type


```
@ElementCollection(fetch = FetchType.LAZY)
private List<Phone> phones;
```

```
Person person = em.find(Person.class, 1L);

person.getPhones(); // tehakse uus päring
```

LAZY vs EAGER

LAZY

- Probleemid: Jõudlus, ühenduse katkemine

LAZY - ühenduse katkemine

```
Person person = em.find(Person.class, 1L);
em.close();

person.getPhones(); // error
```

EAGER

```
@Data
@Entity
public class Person {

    @OneToMany(fetch = FetchType.EAGER)
    private List<Phone> phones;
```

```
Person person = em.find(Person.class, 1L);

person.getPhones(); // loetakse mälust
```

EAGER

- Probleem: paindumatus, ootamatus

```
@Data
@Entity
public class Person {

    @OneToMany(fetch = FetchType.EAGER)
    private List<Phone> phones;
```

```
List<Person> persons = em.createQuery(
        "select p from Person p", Person.class)
        .getResultList();
```

Projekti 9. osa

- Jpa kasutamine
- "hibernate.hbm2ddl.auto" väärtus on "validate"
- Jpa, Spring Mvc, JSR 303, Hsql, Gradle, Spring Boot (valikuliselt).

ORM-i Jõudlusest

Java Persistance API (JPA)

- Java ORM-i standard
- Määrab liidesed ja annotatsioonid: EntityManager, @Entity, @Id, @ManyToOne, ...

JPA vs Hibernate

Jpa puudused

- Paindumatus
- Keerukus
- Harjumatus
- Raportid

Jpa eelised

- Vähendab sõltuvust andmebaasist
- Võimalik kasutada mälupõhist andmebaasi testkeskonnas
- Lihtsustab keerulise andmemudeliga programmi disaini (salvestamise osa)

Millal kasutada?

- Kui on keeruline domeenimudel
- Kui rakendusel on „õpiku arhitektuur“

EntityManagerFactory

- Loomine on kallis operatsioon:
  - Laeb raamistiku
  - Loeb konfiguratsiooni
  - Loob/uuendab/valideerib skeemi
  - Loob ühendused

EntityManager

- Loomine on suhteliselt odav operatsioon

Olemite genereerimine

Native Queries

```
Query query = em.createNativeQuery(
        "TRUNCATE SCHEMA public AND COMMIT");

query.executeUpdate();
```

Nimelised päringud

```
@Entity
@NamedQuery(name="Employee.findAll",
            query="SELECT e FROM Employee e")
public class Employee { ...
```

Nimelised päringud

```
TypedQuery<Employee> query = em.createQuery(
        "select e from Employee e where e.id = :id",
        Employee.class);
```

```
TypedQuery<Employee> namedQuery = em.createNamedQuery(
        "Employee.findAll", Employee.class);

List<Employee> employees = namedQuery.getResultList();
```

Nimelised päringud

- Valideeritakse rakenduse käivitamisel
- Rakendus ei käivitu, kui mõnes päringus viga on

Nimelised päringud

```
@Entity
@NamedQueries({
        @NamedQuery(name="Person.findAll",
            query="SELECT p FROM Person p"),
        @NamedQuery(name="Person.findByName",
            query="SELECT p FROM Person p WHERE p.name = :name"),
        @NamedQuery(...
})
public class Person { ...
```