class: inverse, center, middle

Perzisztencia, adatbázis programozás JPA technológiával

---

class: inverse, center, middle

Tematika

Tematika

- JPA

Források

- Mike Keith, Merrick Schincariol: Pro JPA 2 in Java EE 8: An In-Depth Guide to Java Persistence APIs (3. kiadás)

---

class: inverse, center, middle

Egyszerű mentés és lekérdezés JPA-val

JPA

- JDBC bonyolultsága: leképzés a relációs adatbázis és oo világ között
- Megoldás: keretrendszer biztosítsa konfiguráció alapján
- ORM: Object-Relational Mapping
- Szabvány: JPA
- Implementációi: Hibernate, EclipseLink, OpenJPA
- JDBC-re épül

Entitások

```
@Entity
public class Employee {

    @Id
    private Long id;

    private String name;

    public Employee() {
```

```
    }

    // Getter és setter metódusok
}
```

```
create table employee (id bigint,
  name varchar(255),
    constraint pk_employees primary key (id));
```

Személyre szabás

```
@Entity
@Table(name = "employees")
public class Employee {

    @Id
    private Long id;

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

    // Getter és setter metódusok
}
```

```
create table employees (id bigint,
  emp_name varchar(255),
    constraint pk_employees primary key (id));
```

Azonosítógenerálás

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

```
create table employees (id bigint auto_increment,
  emp_name varchar(255),
    constraint pk_employees primary key (id));
```

EntityManager

- CRUD műveletek

```
Employee employee = new Employee();
employee.setName("John Doe");
entityManager.persist(employee);
```

```
Employee employee = entityManager.find(Employee.class, 1);
employee.setName('Jack Doe');
```

```
Employee employee = entityManager.find(Employee.class, 1);
employee.remove(employee);

entityManager
  .createQuery("select e from Employee e order by e.name",
      Employee.class)
    .getResultList();
```

Tranzakciókezelés

```
entityManager.getTransaction().begin();
entityManager.persist(employee);
entityManager.getTransaction().commit();
```

Persistence unit

```
<persistence version="2.0"
             xmlns="http://java.sun.com/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
              http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
    <persistence-unit name="pu"
                      transaction-type="RESOURCE_LOCAL">
        <class>basic.Employee</class>
        <properties>
            <property name="javax.persistence.jdbc.driver"
value="com.mysql.jdbc.Driver"/>
            <property name="javax.persistence.jdbc.user" value="employees"/>
            <property name="javax.persistence.jdbc.password"
value="employees"/>
            <property name="javax.persistence.jdbc.url"
                value="jdbc:mysql://localhost/employees"/>
            <property name="hibernate.dialect"
                value="org.hibernate.dialect.MariaDB10Dialect"/>
            <property name="hibernate.show_sql" value="true"/>
        </properties>
    </persistence-unit>
</persistence>
```

EntityManagerFactory

```
EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory("pu");
        entityManager = entityManagerFactory.createEntityManager();
```

---

class: inverse, center, middle

Architektúra és integrációs tesztelés

DAO

```
public class EmployeeDao {

    private EntityManagerFactory entityManagerFactory;

    public EmployeeDao(EntityManagerFactory entityManagerFactory) {
        this.entityManagerFactory = entityManagerFactory;
    }

    public void saveEmployee(Employee employee) {
        EntityManager entityManager =
entityManagerFactory.createEntityManager();
        entityManager.getTransaction().begin();
        entityManager.persist(employee);
        entityManager.getTransaction().commit();
        entityManager.close();
    }

}
```

Integrációs tesztelés

```
public class EmployeeServiceTest {

    private EmployeeDao employeeDao;

    @Before
    public void init() {
        EntityManagerFactory entityManagerFactory =
Persistence.createEntityManagerFactory("pu");
        employeeDao = new EmployeeDao(entityManagerFactory);
    }

    @Test
    public void testSaveThenFind() {
        Employee employee = new Employee("John Doe");
        employeeDao.saveEmployee(employee);
        Employee loadedEmployee = employeeDao
            .findEmployeeById(employee.getId());
        assertEquals("John Doe", loadedEmployee.getName());
    }
```

}

---

class: inverse, center, middle

Entitások konfigurálása

Sémagenerálás

- Forrása lehet az annotált entitások vagy szkript
- Célja lehet szkript vagy adatbázis

Generálás adatbázisba

- `none` (alapértelmezett): nincs sémagenerálás
- `create`: csak generálás
- `drop`: séma ürítése
- `drop-and-create`: séma ürítése és generálása

```
<property name="javax.persistence.schema-generation.database.action"
    value="drop-and-create"/>
```

Generálás szkriptbe

```
<property name="javax.persistence.schema-generation.scripts.action"
    value="create"/>
<property name="javax.persistence.schema-generation.scripts.create-target
    value="file:///c:/scripts/create.ddl"/>
```

Annotációk használata

- Field access
- Property access
- Mixed access
  - `@Access(AccessType.FIELD)`, `@Access(AccessType.PROPERTY)`

Tábla konfigurálása

- `@Table` annotáció, `name` attribútum
- `catalog`, `schema`
- `indexes`
- `uniqueConstraints`

Oszlopok konfigurálása

- `@Column` annotáció, `name` attribútum
- `unique`
- `nullable`
- `insertable`, `updateable`, `columnDefinition`
- `table`, lásd secondary table
- `length`
- `precision` és `scale`

Egyszerű típusú mezők

- Primitív típusok és burkolóik
- `String`
- `BigInteger`, `BigDecimal`

Enumeration

- Alapértelmezetten a sorszáma
- `@Enumerated(EnumType.STRING)`

Lob

- `byte[]` és `char[]`, valamint burkolóik
- Szerializálható objektumok
- `@Lob` annotáció

Dátum és időkezelés

- `java.sql.Date`, `java.sql.Time`, és `java.sql.Timestamp`
- `java.util.Date`, `java.util.Calendar`
  - `Temporal` annotáció
- JPA 2.2: `LocalDate`, `LocalTime`, `LocalDateTime`, `OffsetTime`, `OffsetDateTime`

Tranziens mezők

- `transient` kulcsszó
- `@Transient annitáció`

---

class: inverse, center, middle

Elsődleges kulcs

Elsődleges kulcs típusa

- EntityManager műveletek elsődleges kulcs alapján
- Tipikusan egész szám
- Tartózkodjunk a lebegőpontos számoktól és az idő alapú mezőktől

---

Azonosítógenerálás

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
private long id;
```

Azonosítógenerálás táblával

```
@Id
@GeneratedValue(strategy=GenerationType.TABLE)
private long id;
```

```
@TableGenerator(name="Emp_Gen")
@Id
@GeneratedValue(generator="Emp_Gen")
private long id;
```

```
@TableGenerator(name="Emp_Gen",
    table="ID_GEN",
    pkColumnName="GEN_NAME",
    valueColumnName="GEN_VAL")
@Id
@GeneratedValue(generator="Emp_Gen")
private long id;
```

Azonosítógenerálás táblával

```
@TableGenerator(name="Address_Gen",
    table="ID_GEN",
    pkColumnName="GEN_NAME",
    valueColumnName="GEN_VAL",
    pkColumnValue="Addr_Gen",
    initialValue=10000,
    allocationSize=100)
@Id
@GeneratedValue(generator="Address_Gen")
private long id;
```

Azonosítógenerálás szekvenciával

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

@SequenceGenerator(name="Emp_Gen", sequenceName="Emp_Seq")
@Id
@GeneratedValue(generator="Emp_Gen")
private long getId;

CREATE SEQUENCE Emp_Seq
    MINVALUE 1
    START WITH 1
    INCREMENT BY 50;
```

Azonosítógenerálás identity alapján

```
@Id
@GeneratedValue(strategy=GenerationType.IDENTITY)
private long id;

create table employees (id bigint auto_increment,
  emp_name varchar(255),
    constraint pk_employees primary key (id));
```

Összetett azonosítók

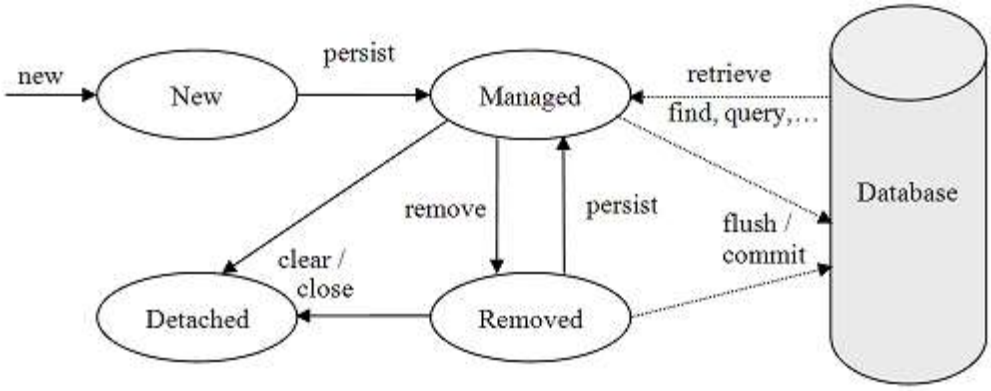
- Mindenképp kell egy külön osztály, mely tartalmazza a mezőket
  - `equals()` és `hashCode()` metódusok szerepe
- Két megoldás
  - Entitásban több mező, `@Id` annotációkkal ellátva, és `@IdClass` annotáció az entitáson
  - Külön osztály típusú mező, `@EmbeddedId` annotációval, a külön osztály `@Embeddable`

---

class: inverse, center, middle

Entitások életciklusa

Életciklus



JPA életciklus

Persistence context fogalma

- Managed állapotú entitások
- Dinamikus, futásidejű
- Vagy tranzakciónyi, vagy tranzakción átnyúló
- Detached
- Vagy explicit módon eltávolításra kerül, pl. `entityManager.clear()` vagy `entityManager.detach()`
- Megszűnik a persistence context
- `entityManager.merge()` metódussal visszacsatolható

Lifecycle események

- `@PrePersist`, `@PostPersist`
- `@PreRemove`, `@PostRemove`
- `@PreUpdate`, `@PostUpdate`
- `@PostLoad`

Flush metódus

- `entityManager.flush()`

---

class: inverse, center, middle

Többértékű attribútumok

Collectionök

- `Collection`, `List`, `Set`, `Map`
- Egyszerű típusok, vagy `@Embeddable` annotációval ellátott osztályok
- `@ElementCollection` annotáció

Egyszerű típus

```
@ElementCollection
private Set<String> nickNames;
```

Példa embeddable

```
@Embeddable
public class VacationEntry {

    private LocalDate startDate;

    private int daysTaken;

    // ...
}

@ElementCollection
private Set<VacationEntry> vacationBookings;
```

Személyre szabás

```
@ElementCollection
@CollectionTable(name="NICKNAMES", joinColumns=@JoinColumn(name="EMP_ID"))
@Column(name="NICKNAME")
private Set<String> nickNames;

@ElementCollection(targetClass=VacationEntry.class)
@CollectionTable(name="VACATIONS", joinColumns=@JoinColumn(name="EMP_ID"))
@AttributeOverride(name="daysTaken", column=@Column(name="DAYS_ABS"))
private Set<VacationEntry> vacationBookings;
```

Map

- Egyszerű és embeddable típusok tetszőleges kombinációja

```
@ElementCollection
@CollectionTable(name="EMP_PHONE")
@MapKeyColumn(name="PHONE_TYPE")
@Column(name="PHONE_NUM")
private Map<String, String> phoneNumbers;
```

Lazy kapcsolat

- Lazy: csak szükség esetén tölti be az attribútumhoz tartozó értékeket
- Felülbírálása az `@ElementCollection` `fetch` attribútumával (`FetchType.EAGER`)
  - Nem javasolt, mert ez statikus

N + 1 probléma

- Egy lekérdezés az entitásra, majd entitásonként a kapcsolódó attribútumokra
- Megoldás: `join fetch`

Lazy kapcsolat

- `LazyInitializationException`, csak Hibernate esetén
- Detach-elt entitáson lazy kapcsolat betöltése
- Megoldás: `join fetch`

---

class: inverse, center, middle

Kapcsolatok

Kapcsolatok tulajdonságai

- Számosság
  - Egy-egy
  - Egy-több
  - Több-több
- Irányítottság
  - Egyirányú
  - Kétirányú

Egyirányú egy-egy kapcsolat

- `@OneToOne` annotáció
- `@JoinColumn` annotáció

Kétirányú egy-egy kapcsolat

- `@OneToOne` annotáció

- `@JoinColumn` annotáció
- `mappedBy` attribútum
- Külön metódus a két irány beállítására

Kétirányú egy-több kapcsolat

- `@OneToMany` annotáció `mappedBy` attribútummal
- `@ManyToOne` annotáció
- Külön metódus a két irány beállítására

---

getReference

- `EntityManager.getReference()`
- Ha a kapcsolathoz be kell tölteni az entitást
- Proxy-t ad vissza

```
Department dept = em.getReference(Department.class, 30);
```

Kaszkádolt műveletek

- Művelet elvégzése az entitáson és a kapcsolódó entitáson is

```
@Entity
public class Employee {
    // ...

    @ManyToOne(cascade=CascadeType.PERSIST)
    Address address;

    // ...
}
```

Orphan removal

- Szülő rekord eltávolításakor árva marad, törölhető (hasonló, mint a `CascadeType.REMOVE` funkcionalitása)
- Ami több: nem csak törléskor, hanem kapcsolat megszűntetésekor is törli a kapcsolt entitást

```
@Entity
public class Employee {
    @Id
    private int id;

    @OneToMany(orphanRemoval = true)
    private List<Evaluation> evals;
```

```
    // ...
}
```

Sorrendezés

- Attribútum alapján: `@OrderBy` annotáció
- Erre kijelölt mező alapján, JPA tartja karban: `@OrderColumn` annotáció

Szülő oldal

- Owner of relationship, owner side
  - Másik oldal: inverse side
- Ahol a `mappedBy` van, az az inverse oldal
- Ahol a join column van, az az owener side

Lazy kapcsolat

- Eager: betölti a kapcsolódó entitást (entitásokat)
- Lazy: csak szükség esetén tölti be a kapcsolódó entitásokat
- `@ElementCollection` esetén is, alapesetben eager
- `@OneToOne` és `@ManyToOne` alapesetben eager
- `@OneToMany` és `@ManyToMany` alapesetben lazy
- Felülbírálása a `fetch` attribútummal
  - `FetchType.EAGER` és `FetchType.LAZY`
  - Nem javasolt, mert ez statikus

N + 1 probléma

- Egy lekérdezés az entitásra, majd entitásonként a kapcsolódó entitásra
- Megoldás: `join fetch`

Lazy kapcsolat

- `LazyInitializationException`, csak Hibernate esetén
- Detach-elt entitáson lazy kapcsolat betöltése
- Megoldás: `join fetch`

Entity graph

- Mi kerüljön betöltésre

- Lekérdezésben hint megadása
  - `javax.persistence.fetchgraph` - entitáshoz tartozó alapértelmezett entity graph-ot nem veszi figyelembe
  - `javax.persistence.loadgraph`
- Alapértelmezett entity graph: alapértelmezett és annotációkkal megadott lazy betöltési tulajdonságok
- Megadása annotációkkal vagy programozottan

Entity graph deklarálása

```
@NamedEntityGraph(name = "graph.Employee.phones",
    attributeNodes = @NamedAttributeNode("phones"),
    subgraphs = {
        @NamedSubgraph(name = "phones",
            attributeNodes = {@NamedAttributeNode("type")})
})
```

Entity graph használata

```
Map hints = new HashMap();
hints.put("javax.persistence.fetchgraph",
    em.getEntityGraph("graph.Employee.phones"));
return em.find(Employee.class, id, hints);
```

---

class: inverse, center, middle

Több-több kapcsolatok

Több-több kapcsolat

- Mindkét oldalon kollekció
- `@ManyToMany`, inverz oldalon `mappedBy`

Több-több kapcsolat

```
@Entity
public class Employee {

    @Id private long id;
    private String name;

    @ManyToMany
```

```
    private Collection<Project> projects;

    // ...
}

@Entity
public class Project {

    @Id private long id;
    private String name;

    @ManyToMany(mappedBy="projects")
    private Collection<Employee> employees;

    // ...
}
```

Join table

```
@ManyToMany
@JoinTable(name="EMP_PROJ",
joinColumns=@JoinColumn(name="EMP_ID"),
inverseJoinColumns=@JoinColumn(name="PROJ_ID"))
private Collection<Project> projects;
```

---

class: inverse, center, middle

Entitások mapekben

Entitás attribútuma alapján

```
@Entity
public class Project {

    @Id
    private int id;

    private String name;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "projects_employees",
        joinColumns = @JoinColumn(name = "project_id"),
        inverseJoinColumns = @JoinColumn(name = "employee_id"))
    @MapKey(name = "cardNumber")
    private Map<String, Employee> employees = new HashMap<>();
```

```
    // ...
}
```

Külön érték alapján

```
@Entity
public class Project {

    @Id
    private int id;

    private String name;

    @ManyToMany(cascade = CascadeType.ALL)
    @JoinTable(name = "projects_employees",
        joinColumns = @JoinColumn(name = "project_id"),
        inverseJoinColumns = @JoinColumn(name = "employee_id"))
    @MapKeyColumn(name = "employee_in_project_id")
    private Map<String, Employee> employees = new HashMap<>();

    // ...
}
```

---

class: inverse, center, middle

Beágyazott objektumok és másodlagos tábla

Beágyazott objektumok

- Egy tábla, több osztály
- Újrafelhasználható
- `@Embeddable` a beágyazható osztályon
- `@Embedded` a beágyazásnál

Konfigurálás újrafelhasználásnál

```
@Entity
public class Employee {

    @Id
    private long id;
```

```
    private String name;

    @Embedded
    @AttributeOverride(name="state", column=@Column(name="PROVINCE")),
    @AttributeOverride(name="zip", column=@Column(name="POSTAL_CODE"))
    private Address address;

    // ...
}
```

Másodlagos tábla

- Egy entitás, több tábla
- `@SecondaryTable` annotáció az entitáson
- `@Column` annotáció a mezőn, `table` attribútummal

Másodlagos tábla

```
@Entity
@Table(name="EMP")
@SecondaryTable(name="EMP_ADDRESS",
    pkJoinColumns=@PrimaryKeyJoinColumn(name="EMP_ID"))
public class Employee {

    @Id
    private int id;

    private String name;

    @Column(table="EMP_ADDRESS")
    private String street;
```

---

class: inverse, center, middle

Öröklődés

Absztrakt entitás

- Ősosztály `abstract` kulcsszóval, nem példányosítható
- Lekérdezésekben szerepelhet (összes leszármazottra vonatkozik)
- Együtt akarjuk kezelni a leszármazott entitásokat

Mapped Superclasses

- `@MappedSuperclass` annotációval
- Perzisztens tulajdonságok megosztása entitások között
- Nem szerepelhet lekérdezésekben
- Nem akarjuk együtt kezelni a leszármazott entitásokat

Öröklődések leképzési stratégiái

- Single-Table Strategy
- Joined Strategy
- Table-per-Concrete-Class Strategy

---

class: inverse, center, middle

Lekérdezések

Lekérdezések

- JP QL nyelv
- Objektumorientált
- Entitásokon és mezőiken
  - Perzisztens mezők
  - Kapcsolati mezők
- Identification variable
- Futás közben generál SQL-t

Path expression példán

```
SELECT e
FROM Employee e
WHERE e.department.name = 'NA42' AND
    e.address.state IN ('NY','CA')
```

Egyszerű lekérdezés

```
List<Employee> employees = entityManager
  .createQuery("select e from Employee e order by e.name",
      Employee.class)
    .getResultList();
```

Egyszerű lekérdezés stream eredménnyel

```
List<String> employees = entityManager
  .createQuery("select e from Employee e order by e.name",
      Employee.class)
    .getResultStream().map(Employee::getName).collect(Collectors.toList());
```

Egy találat

```
Employee employee = entityManager
  .createQuery("select e from Employee e where e.name = 'John Doe'",
      Employee.class)
    .getSingleResult();
```

Ha több rekord, kivétel

Paraméterezett lekérdezés

```
Employee employee = entityManager
  .createQuery("select e from Employee e where e.name = :name",
      Employee.class)
    .setParameter("name", "John Doe")
    .getSingleResult();
```

Criteria API

- Dinamikus lekérdezések

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> c = cb.createQuery(Employee.class);
Root<Employee> emp = c.from(Employee.class);
c.select(emp)
    .where(cb.equal(emp.get("name"), "John Doe"));
```

Metamodel API

- Erősen típusos, karakterláncok mellőzésére
- Canonical metamodel
  - Generálandó

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> c = cb.createQuery(Employee.class);
Root<Employee> emp = c.from(Employee.class);
c.select(emp)
    .where(cb.equal(emp.get(Employee_.name), "John Doe"));
```

Lapozás

```
List<Employee> employees = entityManager
  .createQuery("select e from Employee e where e.name = :name",
      Employee.class)
    .setParameter("name", "John Doe")
    .setFirstResult(150)
    .setMaxResults(50)
    .getSingleResult();
```

Definiálás named query-ként

- Külön annotációként megadva, entitáson
- Induláskor szintaktikai ellenőrzés
- Gyorsabb futás

```
@NamedQuery(name = "findEmployees", query = "select e from Employee e order
by e.name")

List<Employee> employees = entityManager
  .createNamedQuery("findEmployees",
      Employee.class)
    .getResultList();
```

Named query definiálás futásidőben

```
TypeQuery<Employee> q = em.createQuery("select e from Employee e order by
e.name", Employee.class);
entityManagerFactory.addNamedQuery("findEmployees", q);
```

Join

```
SELECT p.number
FROM Employee e JOIN e.phones p
WHERE e.department.name = 'NA42' AND
p.type = 'Cell'
```

Objektum tömb

```
List<Object[]> employees = entityManager
  .createQuery("select e.id, e.name from Employee e order by e.name",
      Employee.class)
    .getResultList();
```

Projection query

```
List<EmployeeData> employees = entityManager
  .createQuery("select new training360.jpa.EmployeeData(e.id, e.name) " +
    " from Employee e order by e.name",
      Employee.class)
    .getResultList();
```

Query timeout

```
TypeQuery<Employee> q = em.createQuery("select e from Employee e order by
e.name", Employee.class);
q.setHint("javax.persistence.query.timeout", 5000);
```

Timeout esetén `QueryTimeoutException`

---

class: inverse, center, middle

Haladó lekérdezések

Feltételek

- Navigáció (`.`)
- Egy operandusú (+, -)
- Szorzás, osztás (*, /)
- Összeadás, kivonás (+, -)
- Összehasonlítás (=, >, >=, <, <=, <>, `[NOT] BETWEEN`, `[NOT] LIKE`, `[NOT] IN`, `IS [NOT] NULL`, `IS [NOT] EMPTY`, `[NOT] MEMBER [OF]`, `[NOT] EXISTS`, `ANY`, `ALL`, `SOME`)
- Logikai operátorok (`AND`, `OR`, `NOT`)

Aggregált lekérdezések

```
SELECT d, COUNT(e), MAX(e.salary), AVG(e.salary)
FROM Department d JOIN d.employees e
GROUP BY d
HAVING COUNT(e) >= 5
```

Allekérdezések

```
SELECT e
FROM Employee e
WHERE e.salary = (SELECT MAX(emp.salary)
    FROM Employee emp)
```

```
SELECT e
FROM Employee e
WHERE EXISTS (SELECT 1
    FROM Phone p
    WHERE p.employee = e AND p.type = 'Cell')
```

Beépített függvények

- ABS, CONCAT, CURRENT_DATE, CURRENT_TIME, CURRENT_TIMESTAMP, INDEX, LENGTH, LOCATE, LOWER, MOD, SIZE, SQRT, SUBSTRING, UPPER, TRIM

Order by

```
SELECT e, d
FROM Employee e JOIN e.department d
ORDER BY d.name, e.name DESC
```

CASE

```
SELECT p.name,
    CASE WHEN TYPE(p) = DesignProject THEN 'Development'
        WHEN TYPE(p) = QualityProject THEN 'QA'
        ELSE 'Non-Development'
    END
FROM Project p
WHERE p.employees IS NOT EMPTY
```

---

class: inverse, center, middle

Bulk műveletek

Bulk update

```
public void assignManager(Department dept, Employee manager) {
    em.createQuery("UPDATE Employee e " +
        "SET e.manager = ?1 " +
        "WHERE e.department = ?2")
    .setParameter(1, manager)
    .setParameter(2, dept)
    .executeUpdate();
}
```

Bulk delete

```
public void removeEmptyProjects() {
    em.createQuery("DELETE FROM Project p " +
            "WHERE p.employees IS EMPTY")
        .executeUpdate();
}
```

---

class: inverse, center, middle

JPA Spring Boottal

Spring Boot JPA-val

- Spring Framework: DI keretrendszer
- Spring Boot: Spring Framework konfigurálása egyszerűen
  - Előre konfigurált függőségek
- Konténer, és komponensek: beanek

JPA

- `DataSource`, `EntityManagerFactory` (persistence context) előre konfigurálása (`application.properties` alapján)
- `persistence.xml` nem szükséges
- Deklaratív tranzakciókezelés

DAO osztály

```
@Repository
public class EmployeeDao {

    @PersistenceContext
    private EntityManager entityManager;

    @Transactional
    public void saveEmployee(Employee employee) {
        entityManager.persist(employee);
    }

    public Employee findEmployeeById(long id) {
        return entityManager.find(Employee.class, id);
    }
}
```

Teszteset

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class EmployeeDaoTest {

    @Autowired
    private EmployeeDao employeeDao;

    @Test
    public void saveAndFind() {
        Employee employee = new Employee("John Doe");
        employeeDao.saveEmployee(employee);

        Employee anotherEmployee = employeeDao
            .findEmployeeById(employee.getId());
        assertEquals("John Doe", anotherEmployee.getName());
    }

}
```

Spring Data JPA

- Egyszerűbbé teszi a perzisztens réteg implementálását
- Tipikusan CRUD műveletek támogatására, olyan gyakori igények megvalósításával, mint a rendezés és a lapozás
- Interfész alapján repository implementáció generálás
- Ismétlődő fejlesztési feladatok redukálása, *boilerplate* kódok csökkentése

---

Maven függőség

```
<dependency>
    <groupId>org.springframework.data</groupId>
    <artifactId>spring-data-jpa</artifactId>
    <version>2.1.2.RELEASE</version>
</dependency>
```

Konfiguráció

- `@Configuration` annotációval ellátott osztályban

```
@EnableJpaRepositories
```

CrudRepository

```
import org.springframework.data.repository.CrudRepository;

public interface EmployeeRepository extends CrudRepository<Employee, Long> {
}
```

A következő metódusokat definiálja: `save(Employee)`, `saveAll(Iterable<Employee>)`, `findById(Long)`, `existsById(Long)`, `findAll()`, `findAllById(Iterable<Long>)`, `count()`, `deleteById(Long)`, `delete(Employee)`, `deleteAll()`, `deleteAll(Iterable<Employee>)`

Teszt eset CrudRepository-ra

Példa teszt eset:

```
@Test
public void testSaveThenFindAll() {
    employeeRepository.save(new Employee("John Doe"));

    Iterable<Employee> employees = employeeRepository.findAll();
    assertEquals(List.of("John Doe"),
StreamSupport.stream(employees.spliterator(), false)
      .collect(Collectors.toList()));
}
```

PagingAndSortingRepository

```
public interface PagingAndSortingRepository<T, ID extends Serializable>
  extends CrudRepository<T, ID> {

  Iterable<T> findAll(Sort sort);

  Page<T> findAll(Pageable pageable);
}
```

```
Sort.by("name")
Sort.by("name").descending()
PageRequest.of(1, 20)
PageRequest.of(1, 20, Sort.by("name"))
```

Page

- `int getNumberOfElements()`
- `List<T> getContent()`
- `int getTotalPages()`

- `long getTotalElements()`
- `boolean isFirst()`, `boolean isLast()`, `boolean hasNext()`, `boolean hasPrevious()`, `Pageable getPageable()`, `Pageable nextPageable()`, `Pageable previousPageable()`

Lekérdező metódusok

- Query Creation

```
interface EmployeeRepository extends Repository<Employee, Long> {

  List<Employee> findByName(String name);

  List<Employee> findDistinctEmployeeByNameOrEmail(String name, String
email);

  List<Employee> findByNameIgnoreCase(String name);

  List<Employee> findByNameOrderByNameAsc(String name);

}
```

- Visszatérés: `List<T>`, `Iterable<T>`, `Stream<T>`
- `Stream<T>` lezárandó, érdemes try-with-resources szerkezetben

Metódus definiálása JPA query-vel

Query annotáció használatával

```
@Query("select e from Employee e where length(e.name) = :nameLength")
Iterable<Employee> findByNameLength(@Param("nameLength") int nameLength);
```

- Megadható count query `countQuery` paraméterként
- Megadható native query `nativeQuery = true` paraméter esetén

Saját implementáció - interfész

Saját interfész:

```
public interface CustomizedEmployeeRepository {

    List<Employee> findByNameStartingWithAsList(String namePrefix);
}
```

Saját implementáció - implementáció

```
public class CustomizedEmployeeRepositoryImpl implements
CustomizedEmployeeRepository {

    @Autowired
    private EntityManager entityManager;

    @Override
    public List<Employee> findByNameStartingWithAsList(String namePrefix) {
        return entityManager
            .createQuery(
                "select e from Employee e where e.name like :namePrefix",
                Employee.class)
            .setParameter("namePrefix", namePrefix + "%").getResultList();
    }
}
```

A saját implementáció neve kötelezően a repository interfész neve az `Impl` posztfixszel

---

Saját implementáció - Repository interfész

Interfész kiterjessze az új interfészt is:

```
public interface EmployeeRepository
    extends CrudRepository<Employee, Long>, CustomizedEmployeeRepository {
        // ...
}
```

---

class: inverse, center, middle

JPA Java EE-vel

Java EE

- Szabvány nagyvállalati alkalmazásfejlesztésre
- Java SE-re épül
- Plusz igények:
- Perzisztencia, tranzakciókezelés
- Távoli elérés
- Párhuzamos kiszolgálás
- Biztonság, HA, scalability
- Implementációi: alkalmazásszerverek

- Konténer és komponensek: EJB-k

JPA a Java EE-n belül

- `DataSource` JNDI-ben, alkalmazásszerverben konfigurált
- `EntityManagerFactory` (persistence context) előre konfigurálása `persistence.xml` alapján
- Deklaratív és programozott tranzakciókezelés

DAO

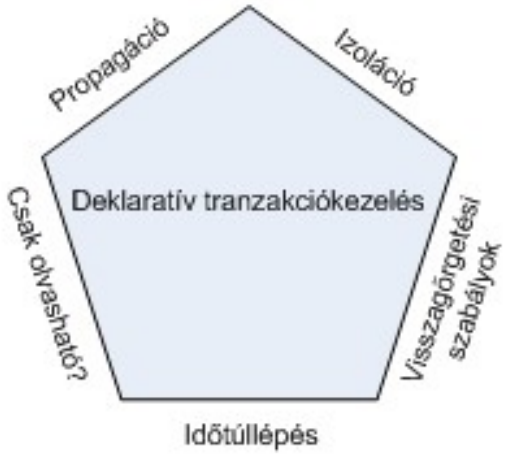
```
@Stateless
public class EmployeeDao {

    @PersistenceContext
    private EntityManager entityManager;

    @Transactional
    public void saveEmployee(Employee employee) {
        entityManager.persist(employee);
    }

    public List<Employee> listEmployees() {
        return entityManager
            .createQuery("select e from Employee e", Employee.class)
                 .getResultList();
    }
}
```

---

class: inverse, center, middle

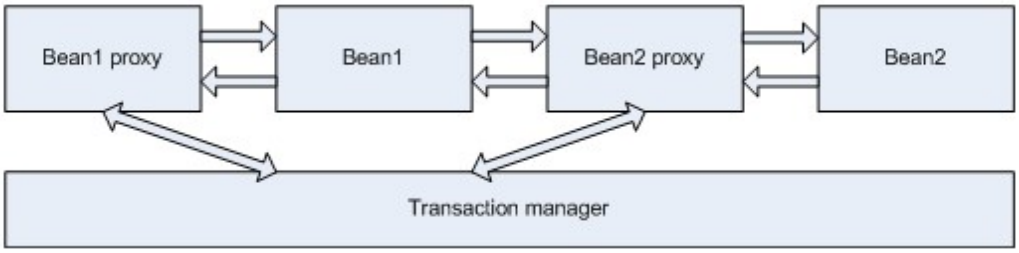
Deklaratív tranzakciókezelés

Tranzakciókezelés



Tranzakciókezelés

Propagáció



Propagáció

Propagációs tulajdonságok

- `REQUIRED` (default): ha nincs tranzakció, indít egyet, ha van csatlakozik hozzá
- `REQUIRES_NEW`: mindenképp új tranzakciót indít
- `SUPPORTS`: ha van tranzakció, abban fut, ha nincs, nem indít újat
- `MANDATORY`: ha van tranzakció, abban fut, ha nincs, kivételt dob
- `NOT_SUPPORTED`: ha van tranzakció, a tranzakciót felfüggeszti, ha nincs, nem indít újat
- `NEVER`: ha van tranzakció, kivételt dob, ha nincs, nem indít újat

---

Izoláció

- Izolációs problémák:
  - dirty read
  - non-repetable read
  - phantom read
- Izolációs szintek:
  - read uncommitted
  - read commited
  - repeatable read
  - serializable

Visszagörgetési szabályok

- Kivételekre lehet megadni, hogy melyik esetén történjen rollback
- Rollbackre explicit módon megjelölni
- Konténer dönt a commitról vagy rollbackről

Timeout

- Timeout esetén kivétel

Csak olvasható

- Spring esetén további optimalizációkat tud elvégezni, cache-eléssel kapcsolatos