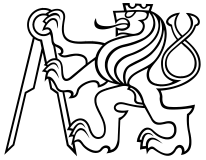
WA2

JPA

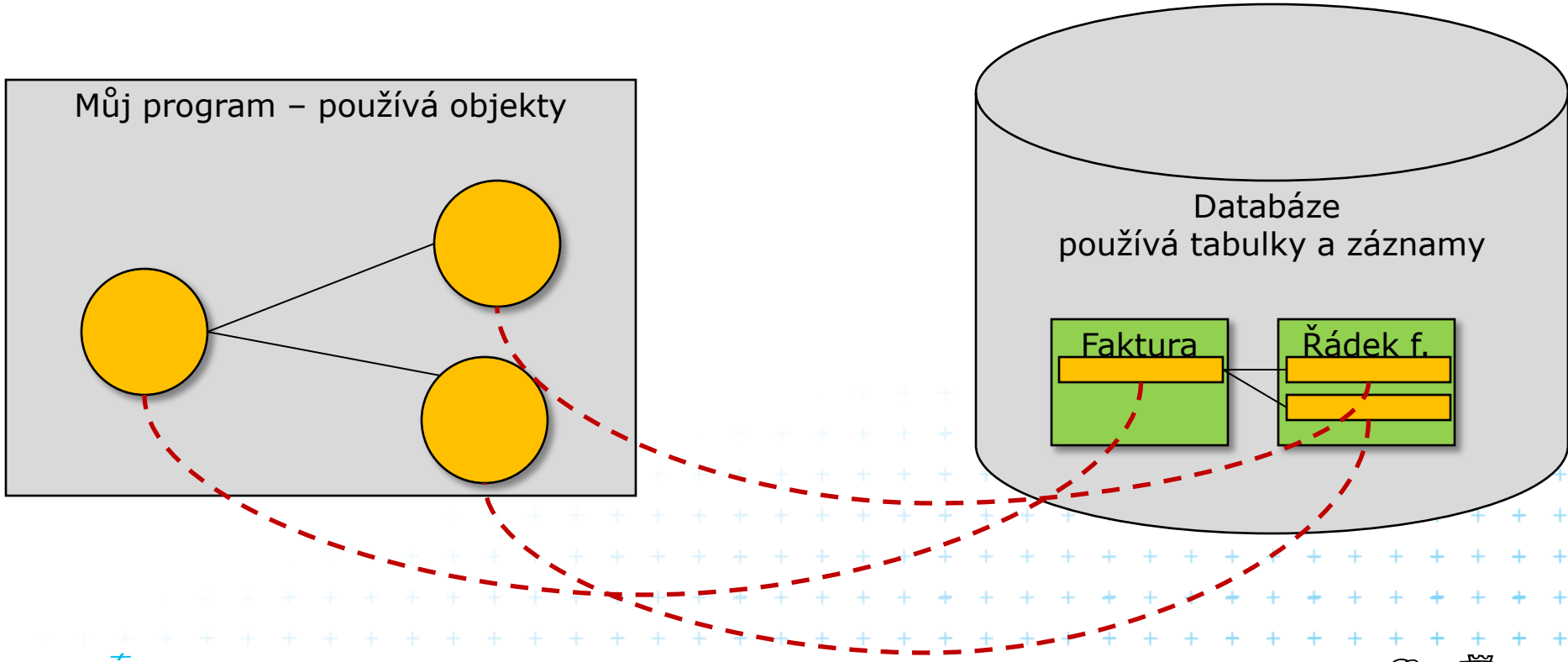
Martin Klíma

DCGI



JPA – Java Persistence API

- Jedná se o API k systému ORM
- ORM = Object – Relational Mapping



ORM – proč je to potřeba?

- Objektové programování rádo pracuje s objekty
- Existují objektové databáze, které toto řeší
  - problém s výkonem
  - problém se standardizací
- Komerční řešení většinou založena na RELAČNÍCH DB
- Přímá práce s DB např. pomocí JDBC:
  - je nepohodlná
  - zvyšuje chybovost
  - snižuje přenositelnost kódu
  - míchá dvě logické úrovně

ORM v Java EE

Do J2EE 1.4 existovaly speciální Entity Beans

- sada tříd a rozhraní
- hodně konfiguračních souborů
- značně nepohodlné a složité

- ale …
- perzistence plně v kompetenci kontejneru
- řeší transakce
- load balancing

Je něco lepšího?

… ano

Hibernate

- přišlo s jednoduchým konfiguračním XML souborem, který mapuje POJO třídy na rel. DB

JPA

- jako modernější reakce na Hibernate dělá prakticky totéž, používá anotace

Základní myšlenka

- Používáme klasické POJO objekty
- Automatické mapování pomocí jmen
- Anotují se pouze výjimky
- Kontejner se postará o injekci zdrojů
- V objektech funguje dědičnost

Entity Manager

- Stará se o práci s entitami, připojení
- Získat ho můžeme buď injekcí

```
@PersistenceContext(unitName="school_persistence_unit")
    private EntityManager em;

//…následuje práce s em
```

- nebo pomocí factory

```
javax.persistence.EntityManager em;
        em = javax.persistence.
       Persistence.createEntityManagerFactory(
   " school_persistence_unit ").createEntityManager();

//…následuje práce s em
```


Entity manager - konfigurace

- Konfigurace je uložena v souboru persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" 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_1_0.xsd">
  <persistence-unit name="school_persistence_unit" transaction-type="JTA">
    <provider>oracle.toplink.essentials.PersistenceProvider</provider>
    <jta-data-source>jdbc/school</jta-data-source>
    <properties>
      <property name="toplink.ddl-generation" value="drop-and-create-tables"/>
    </properties>
  </persistence-unit>
</persistence>
```

název PU bude použit při injection

provider = implementace JPA

JNDI jméno

strategie generování datového úložiště

- Může být definováno více PU v jednom souboru
- Provider – implementace
  - Hibernate, Oracle Toplink, OpenJPA

Entity Manager

persist(Object entity)

uloží entitu do databáze

refresh(Object entity)

znovunačte z DB

merge(T entity)

připojí objekt do perzistentního kontextu

remove(Object entity)

smaže z databáze

find(Class<T>entityClass, Object primaryKey)

vyhledá typ T pomocí jeho primárního klíče

flush()

zapíše do DB

create*Query(String sql, …)

dotaz do db

Ukázka kódu

Zacházej s ní jako s entitou

Klasická POJO třída

```
@Entity
public class TeacherEntity implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String firstName;
    private String lastName;
    @OneToMany(mappedBy = "supervisor")
    private Collection<StudentEntity> students;

    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;   }
    public Collection<StudentEntity> getStudents() {
        return students;
    }
}
```

Entita potřebuje identifikátor

Vlastnosti a k nim gettery a settery

Kolekce jiných entit (představuje relaci v ERDB

Jeden učitel může supervizovat více žáků

Ukázka kódu pokr.

```
@Stateless
public class SchoolSessionBean implements SchoolSessionBeanRemote,
SchoolSessionBeanLocal {

    @PersistenceContext(unitName = "school_persistence_unit")
    private EntityManager em;

    public StudentEntity addStudent(final String firstName, final String lastName) {

        StudentEntity newStudent = new StudentEntity();
        newStudent.setFirstName(firstName);
        newStudent.setLastName(lastName);

        em.persist(newStudent);
        return newStudent;
    }
```

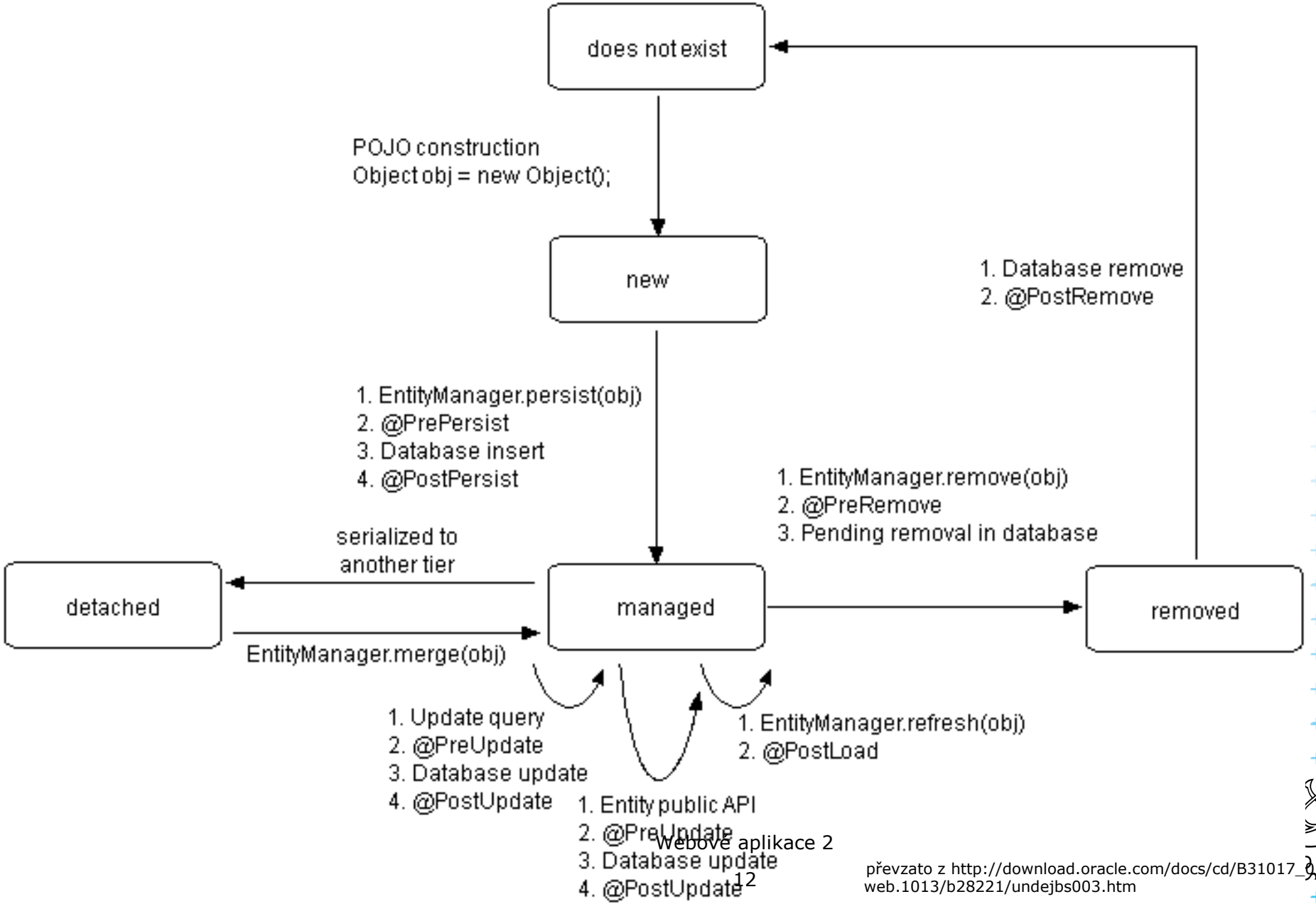
Session EJB

Injekce EM

Vyrobíme novou instanci

Necháme jí zapsat do databáze

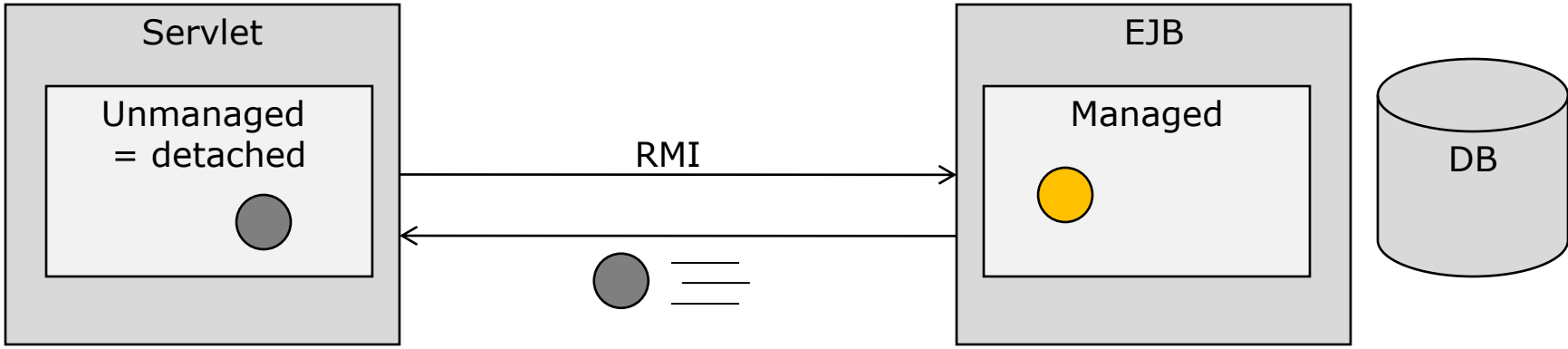
Životní cyklus Entity



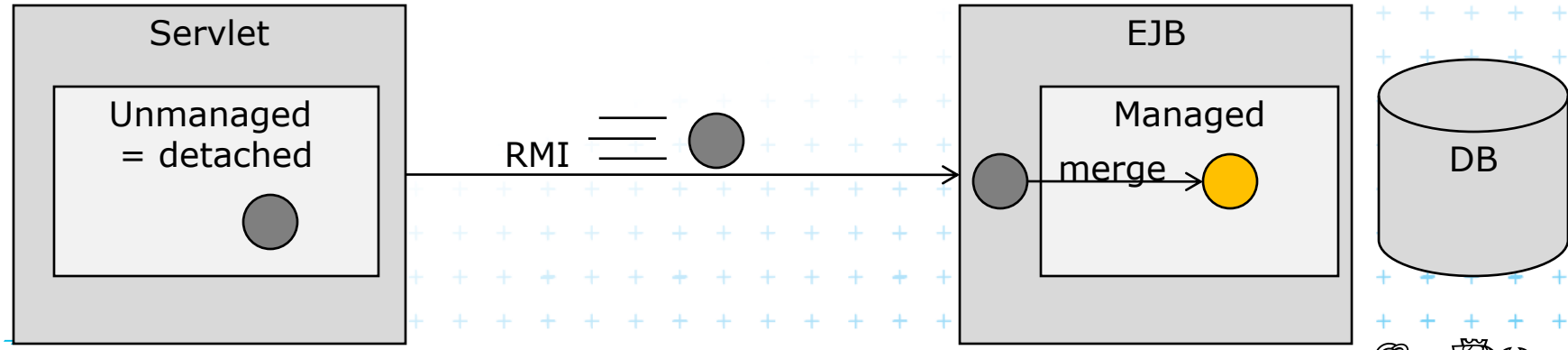
převzato z http://download.oracle.com/docs/cd/B31017_01/web.1013/b28221/undejbs003.htm

merge

- typický příklad



- Servlet změní nějakou vlastnost Entity, potom merge



Ukázka kódu

```
@Stateless
public class SchoolSessionBean implements SchoolSessionBeanRemote,
SchoolSessionBeanLocal {

    @PersistenceContext(unitName = "school_persistence_unit")
    private EntityManager em;

    public void setSupervisor(final StudentEntity student, final TeacherEntity supervisor) {
        student.setSupervisor(supervisor);
        em.merge(student);
    }
}
```

Entita

- musí mít prázdný *public* nebo *protected* konstruktor
- musí mít anotaci *javax.persistence.Entity*
- nesmí být final a ani její metody a proměnné nesmí být final
  - to proto, že kontejner ji ještě obrábí pomocí dědičnosti
- pokud bude serializovaná (např. RMI), musí implementovat interface *Serializable*
- musí mít primární klíč @Id

- anotovat lze proměnné třídy instance nebo její metody
  - nelze to kombinovat

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

```
@Id
  @GeneratedValue(strategy = GenerationType.AUTO)
  public Long getId() {
     return id;
  }
```

Vztahy mezi entitami

Jedná se o klasické relačně databázové vztahy

1:N, 1:1, M:N

Vlastník relace je vždy ten, kdo u sebe drží klíč druhé entity,

např. 1:N

u M:N může být vlastníkem kterákoli strana

Unidirectional a bidirectional

- Unidirectional
  - jen jedna strana (vlastník) ví o té druhé

```
@ManyToOne ()
```

- Bidirectional
  - obě strany o sobě vzájemně ví
  - jedna strana je vlastníkem

```
@ManyToOne ()
```

  - druhá o té druhé ví

```
@OneToMany(mappedBy = "supervisor")
```

Příklad

```
public void setSupervisor(final StudentEntity
student, final TeacherEntity supervisor) {
        student.setSupervisor(supervisor);
        em.merge(student);
        // pozor, toto ne!
        //supervisor.getStudents().add(student);
        //em.merge(supervisor);
    }
```

Příklad @ManyToMany

Subject

```
@Entity
public class SubjectEntity implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;

    @ManyToMany
    private Collection<TeacherEntity> teachers;

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


...

    public Collection<TeacherEntity> getTeachers() {
          return teachers;   }
    public void setTeachers(Collection<TeacherEntity>
teachers) {    this.teachers = teachers;   }
}
```

Vlastník

Teacher

```
@Entity
public class TeacherEntity implements Serializable {
    @ManyToMany(mappedBy = "teachers")
    private List<SubjectEntity> subjectEntitys;
    private static final long serialVersionUID = 1L;
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @ManyToMany(mappedBy = "teachers")
    private List<SubjectEntity> subjectEntitys;

    public Long getId() { return id; }

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

...
    public Collection<StudentEntity> getStudents()
       return students;  }
    public List<SubjectEntity> getSubjectEntitys() {
        return subjectEntitys;   }
}
```

Informovaný

```
public void LinkTeacherSubject (SubjectEntity subject, TeacherEntity teacher){
        subject.getTeachers().add(teacher);
        em.merge(subject);
        // ale ne:
        // teacher.getSubjectEntitys().add(subject);
        // em.merge(teacher);
    }
```

Provázání

Anotace - detail

- Mnoho anotací má doplňkové parametry, jsou dosti logické, bližší popis např. zde:
  http://www.oracle.com/technology/products/ias/toplink/jpa/resources/toplink-jpa-annotations.html
- @OneToMany(cascade="ALL")
  - ALL - all cascading operations performed on the source entity are cascaded to the target of the association.
  - MERGE - if the source entity is merged, the merge is cascaded to the target of the association.
  - PERSIST - if the source entity is persisted, the persist is cascaded to the target of the association.
  - REFRESH - if the source entity is refreshed, the refresh is cascaded to the target of the association.
  - REMOVE - if the source entity is removed, the target of the association is also removed.
- @JoinColumn
  - name, referencedColumnName, unique, nullable, insertable, columnDefinition, table
- @Transient
  - položky, které se nebudou mapovat do DB

Anotace pokr.

- @GeneratedValue(strategy=
  - Sequence – sekvence
  - AUTO – JPA vybere nejlepší strategii samo
  - TABLE – bude se používat pomocná tabulka, aby se zajistila jedinečnost
  - IDENTITY – jednoznačný identifikáror zajistí DB

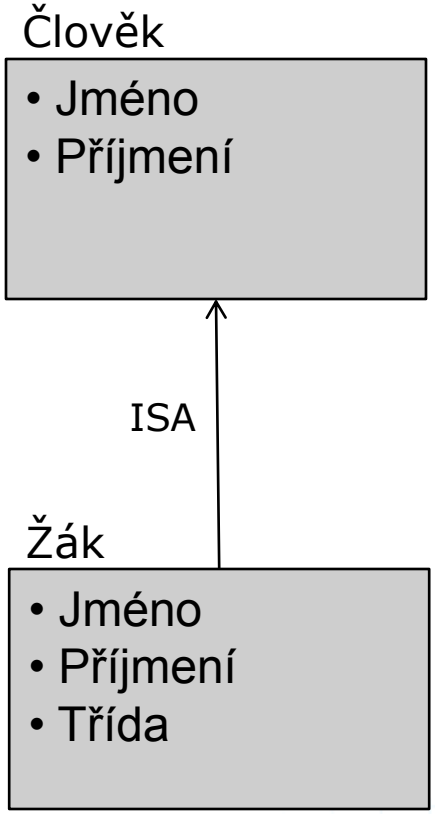
- @LOB
  - označuje vlastnost, která se bude mapovat na BLOB

Ukázka zpětného vygenerování entity StudentEntity

```
@Entity
@Table(name = "STUDENTENTITY")
@NamedQueries({
    @NamedQuery(name = "Studententity.findAll", query = "SELECT s FROM Studententity s"),
    @NamedQuery(name = "Studententity.findById", query = "SELECT s FROM Studententity s WHERE s.id = :id"),
    @NamedQuery(name = "Studententity.findByLastname", query = "SELECT s FROM Studententity s WHERE s.lastname =
:lastname"),
    @NamedQuery(name = "Studententity.findByFirstname", query = "SELECT s FROM Studententity s WHERE s.firstname =
:firstname")})
public class Studententity implements Serializable {
    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @Column(name = "ID")
    private Long id;
    @Column(name = "LASTNAME")
    private String lastname;
    @Column(name = "FIRSTNAME")
    private String firstname;
    @JoinColumn(name = "SUPERVISOR_ID", referencedColumnName = "ID")
    @ManyToOne
    private Teacherentity supervisorId;

    public Studententity() {    }

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

    public Long getId() {    return id;    }

…

}
```

Dědičnost = ISA vztah

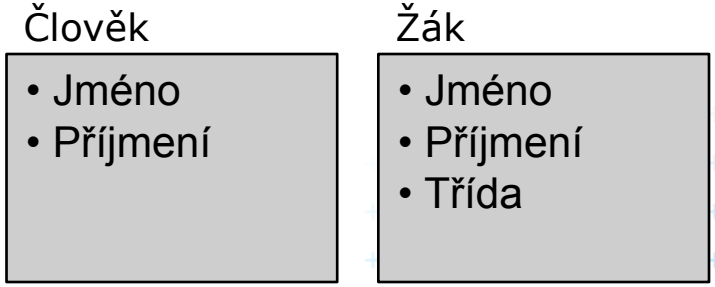
Logická úroveň



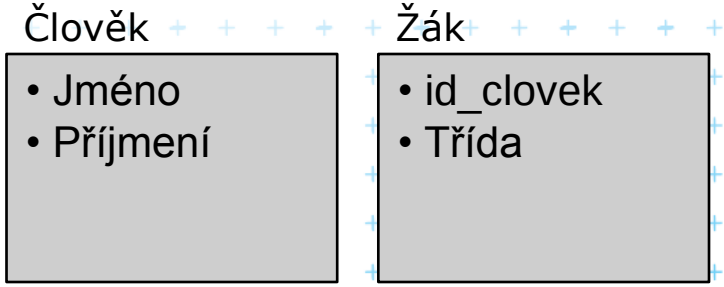
Fyzická úroveň

SINGLE_TABLE
vše v jedné tabulce, rozlišující sloupec

TABLE_PER_CLASS
každá entita má vlastní tabulku s celou sadou atributů



JOINED
hlavní tab. má základní attrib., ostastní se s ní spojují (slabé entity)



Dědičnost – implementace v JPA

```
@Entity
@Inheritance(strategy=InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name="TEACHER_TYPE", discriminatorType=DiscriminatorType.STRING,
length=4)
@DiscriminatorValue(value="FULL")
public class TeacherEntity implements Serializable {
…
}
```

```
@Entity
@DiscriminatorValue(value="PART")
public class PartTimeTeacherEntity extends TeacherEntity implements Serializable {
…
}
```

#	ID	TEACHER_TYPE	LASTNAME	FIRSTNAME	PARTTIME
1	2	FULL	Bručoun	Aleš	<NULL>
2	3	PART	Labuda	Petr	0.5

Dotazovací jazyk

- Jazyk velmi podobný SQL
- Java Persistence query language

```
public Collection<TeacherEntity> findTeacherByLastName(final String lastName) {
    return em.createQuery(
        "SELECT t FROM TeacherEntity t WHERE t.lastName LIKE :paramName")
        .setParameter("paramName", lastName)
        .setMaxResults(10).getResultList();
}
```

Named Query

připravují se pomocí anotace

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

použití

```
public Collection<TeacherEntity> findTeacherByFirstname(final String firstName){
      return em.createNamedQuery("Teacherentity.findByFirstname")
            .setParameter("firstname", firstName)
            .getResultList();
   }
```