Tvorba pokročilých webových aplikací v Javě

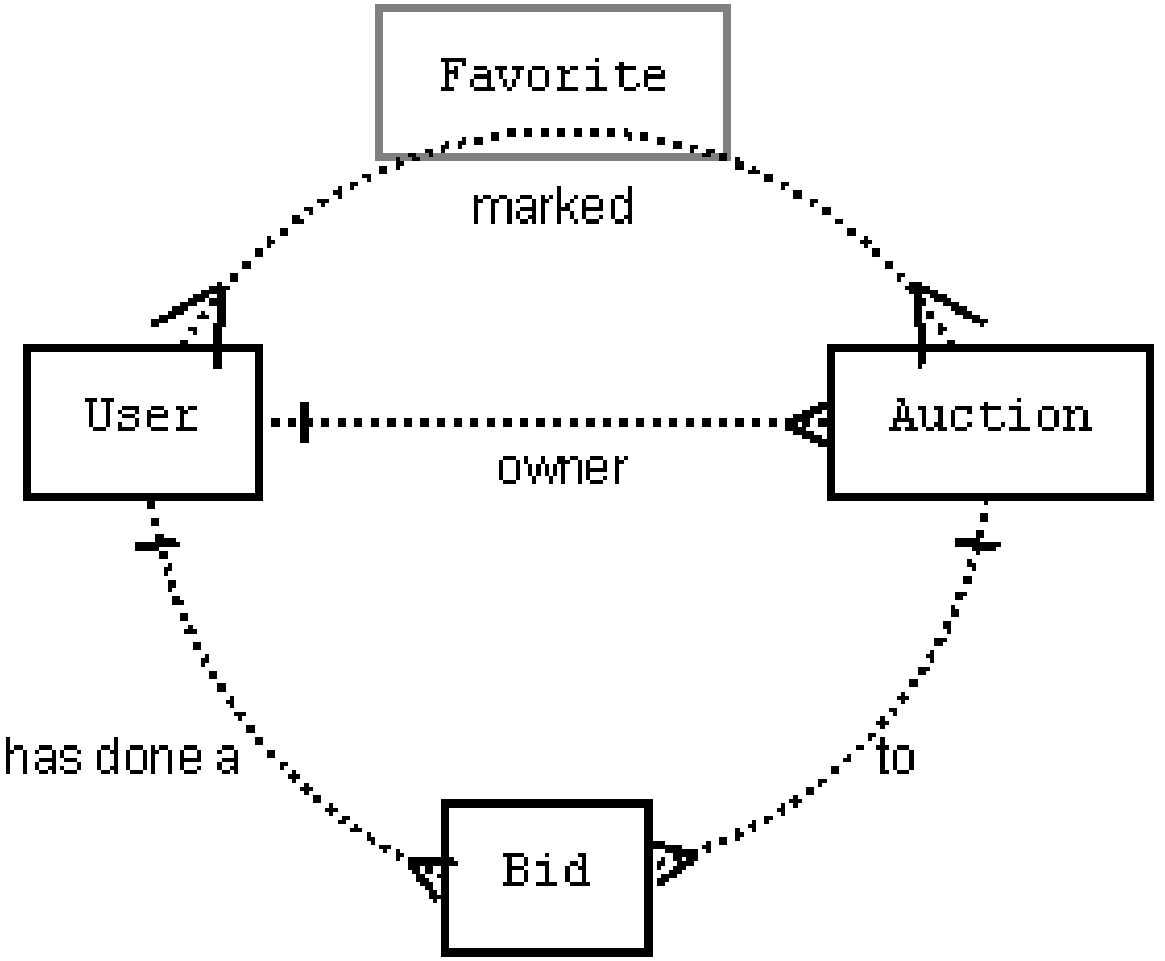
Část 3. - JPA – Java Persistence API (ukládání objektových dat do databáze)

Ondřej Žižka

Říjen 2010

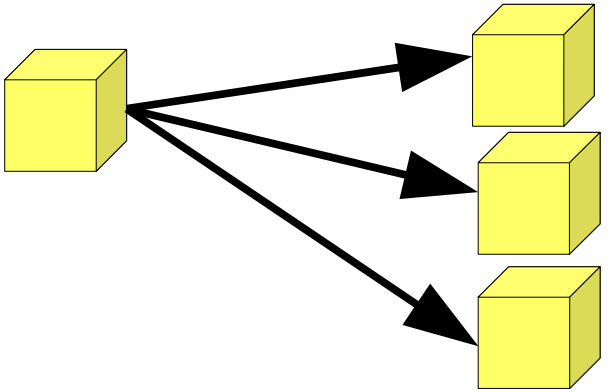
Model našeho projektu

```
public class Auction {
    String title;
    User owner;
    List<Bid> bids;
    ...
}

public class User {
    String name;
    ...
}

public class Bid {
    User bidder;
    int amount;
}
```

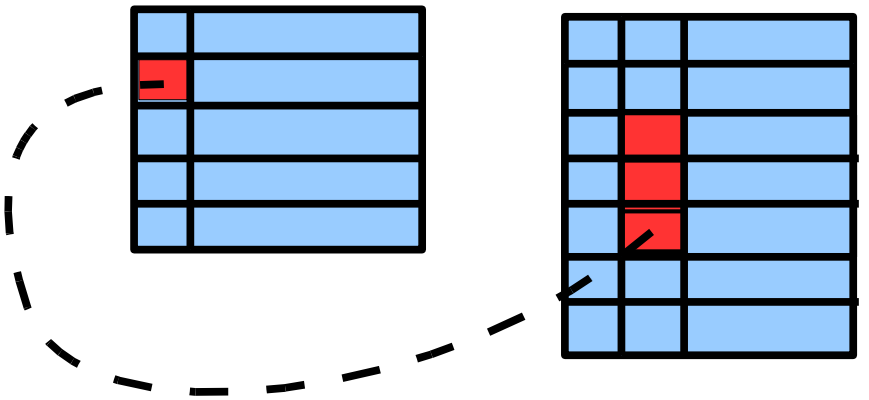


Objekty vs. relační DB: konflikt paradigmat

```
public class Faktura {
    List<Polozka> polozky;
}

public class Polozka {
    int cena;
    int pocet;
}
```



```
CREATE TABLE faktury (
   id INT PK
)
CREATE TABLE polozky (
   id INT PK,
   id_faktura INT FK
)
```



```
SELECT * FROM polozky p
LEFT JOIN faktury f
ON p.id_faktura = f.id;
```

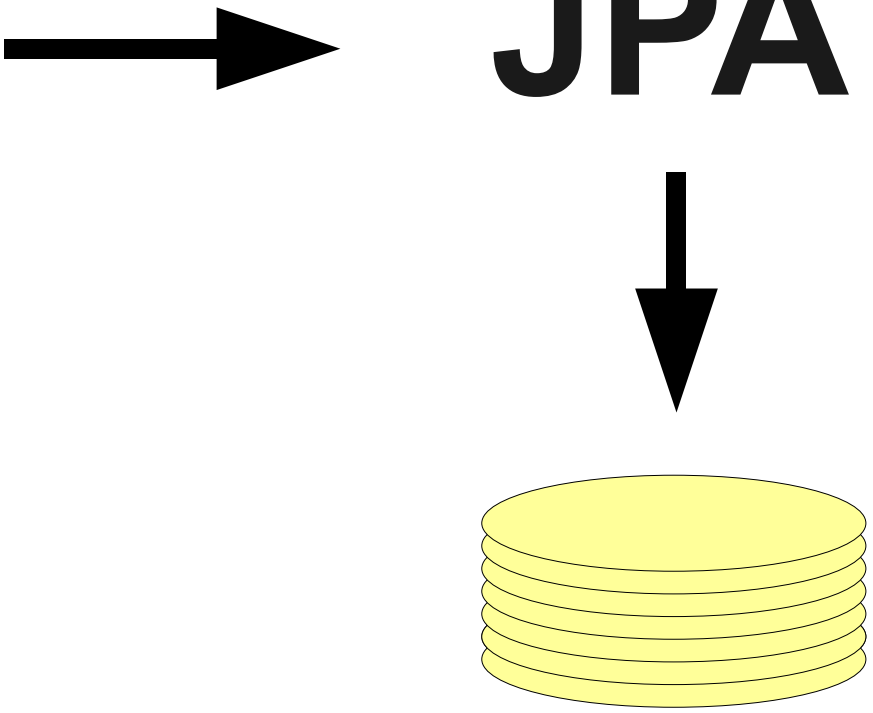
Objektově-relační mapování (ORM)

```
public class Auction {
    String name;
    User owner;
    List<Bid> bids;
}

public class User {
    String name;
}

public class Bid {
    User bidder;
    int amount;
}
```

```
CREATE TABLE auctions (
  id     INT UNSIGNED PK,
  name   VARCHAR(200)
  owner  INT UNSIGNED KEY
);

CREATE TABLE users (
  id    INT UNSIGNED PK,
  name VARCHAR(200)
);

CREATE TABLE bids (
  id    INT UNSIGNED PK,
  id_user INT UNSIGNED FK,
  id_auction INT UNSIGNED FK,
  when DATETIME
);
```

Objektově-relační mapování (ORM)

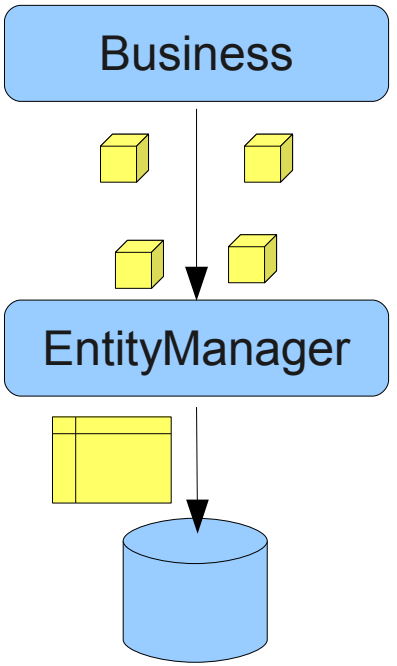
```
@Entity
@Table(name="auctions")
public class Auction {
    @Id long id;
    String name;
    @ManyToOne
    User owner;
    @OneToMany
    List<Bid> bids;
}
@Entity
public class User {
    @Id long id;
    String name;
}
@Entity @Table(name="bids")
public class Bid {
    @Id long id;
    @ManyToOne
    User bidder;
    int amount;
}
```



Java Persistence API (JPA)

- Více implementací
  - Nejpoužívanější je Hibernate - www.hibernate.org
- Snaha standardizovat
- → podmnožina průniku hlavních implementací

- Obvyklá architektura:
  - *Entity* jsou objektová data
  - Business vrstva volá metody *DAO tříd*
  - DAO třídy volají metody EntityManageru



JPA – Entity

- Entita = obyč java třída s JPA anotacemi:

```
@Entity
@Table( name="users" )
public class User {

   public User(){};

   @Id long id;

   @Column( name="name" )
   private String name;
   public String getName();
   public void setName( String name );

   // Pro Set<User> a merge(detachedUser)
   public int hashCode(){ ... }
   public boolean equals(){ ... }
}
```

JPA – ukládání, načtení

- DAO třídy: Nízkoúrovňová práce s objekty – vytváření, změna, ...

```
public class UserDAO {
    @PersistenceContext  // dodá JBoss AS; @Inject
    EntityManager em;

    public void create( String name ){
        em.persist( new User( name ) );
    }

    public User getByID( long id ){
        return em.find( User.class, id );
    }
}
```

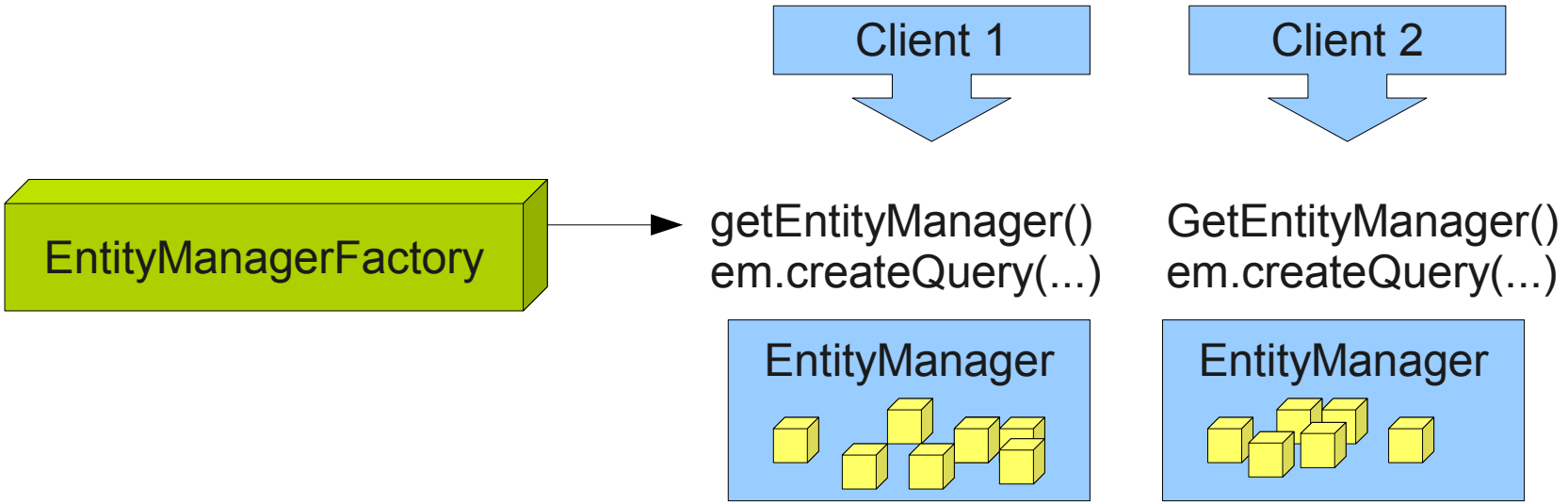
JPA – změna, smazání

- Pozor u **`merge()`** - vrací jiný objekt.

```
public class UserDAO {
    ...

    public User update( User user ){
        return em.merge( user );
    }

    public void remove( User user ){
        em.remove( user );
    }
}
```

JPA – EntityManager

- Co je EntityManager?
  - Schraňuje všechny Entity načtené přes něj.
  - Každý klient / požadavek má obvykle vlastní EM
  - V jednom EM se “stejné” entity vyskytují jen jednou.
    - Ale v různých EM mohou být dvě instance stejné Entity.
  - Při `em.close()` svoje kopie entit uvolní.



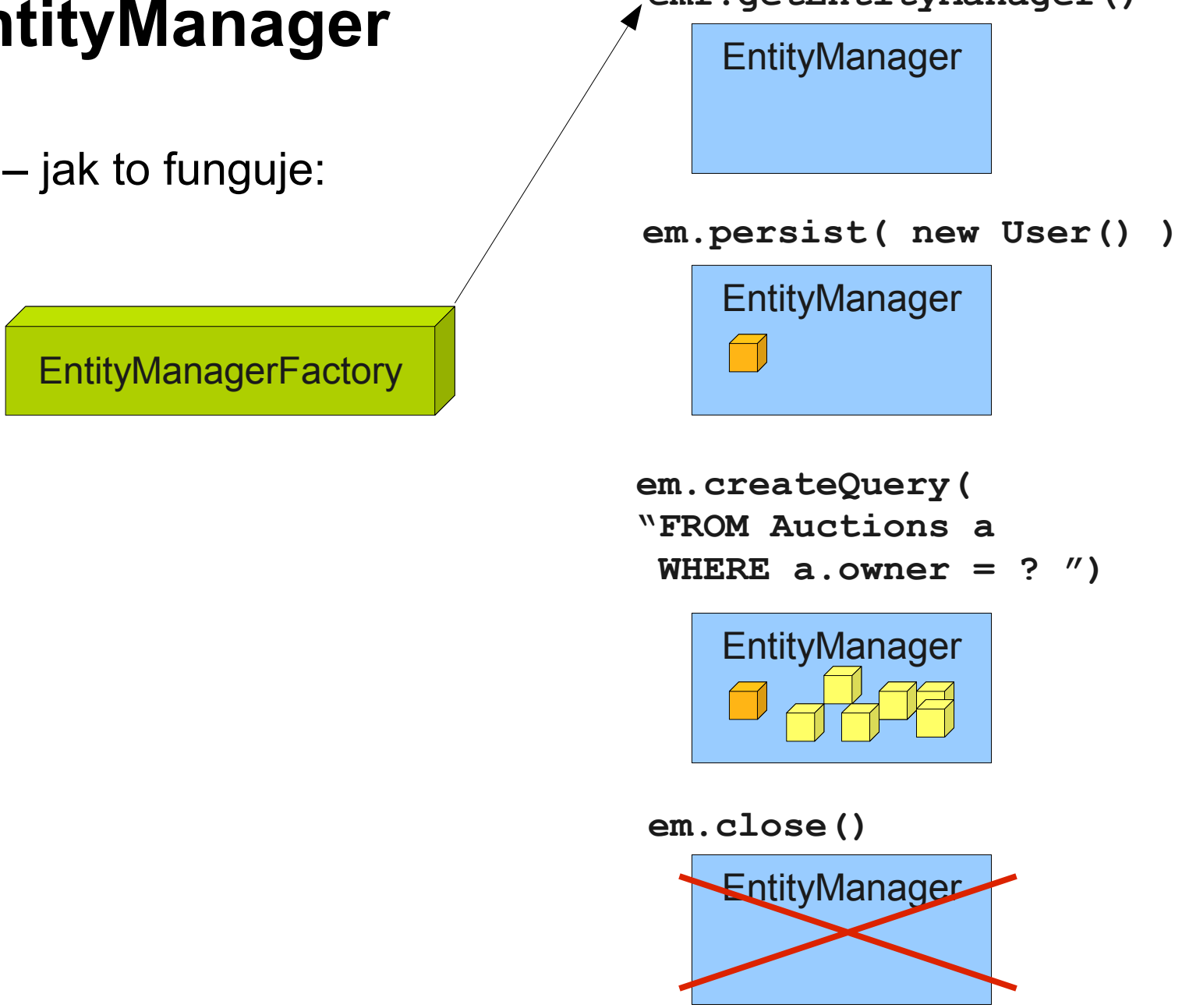
JPA – EntityManager

- Co je EntityManager?
  - Schraňuje všechny Entity načtené přes něj.
  - Každý klient / požadavek má obvykle vlastní EM
  - V jednom EM se “stejné” entity vyskytují jen jednou.
    - Ale v různých EM mohou být dvě instance stejné Entity.
  - Při `em.close()` svoje kopie entit uvolní.
- Takto to vypadá, když nepoužijete aplikační server:

```
public void createAuction( Auction au ){
    EntityManager em = emf.getEntityManager();
    EntityTransaction tx = em.getTransaction();
    tx.begin();
    em.persist( au );
    tx.commit();
    em.close();
}
```

JPA – EntityManager

- Ještě jednou – jak to funguje:



JPA – přínos JBoss Application Serveru

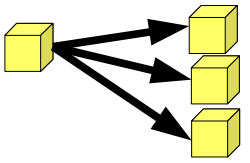
- S JBoss AS:

```
@Inject EntityManager em;

public void createAuction( Auction au ){
  em.persist( au );
}
```

- Takto to vypadá, když nepoužijete aplikační server:

```
public void createAuction( Auction au ){
   EntityManager em = emf.getEntityManager();
   EntityTransaction tx = em.getTransaction();
   tx.begin();
   em.persist( au );
   tx.commit();
   em.close();
}
```

JPA – jednosměrná relace 1:N



- “Owning entity” - vlastník vztahu

```
@Entity public class Faktura {
    @Id long id;
    @OneToMany(cascade=CascadeType.ALL)
    @JoinColumn(name="id_faktura")
    Set<Polozka> polozky = new HashSet();
}
```

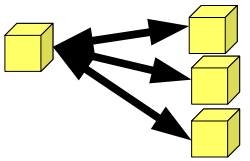
- Druhá entita je při jednosměrném vztahu nedotčená.

```
@Entity public class Polozka {
    @Id long id;
    int pocet;
    int cenaKus; // + getry, setry
    int getCenaCelkem(){ return pocet * cenaKus }
}
```

Viz http://download.oracle.com/javaee/6/api/javax/persistence/OneToMany.html

JPA – obousměrná relace 1:N

Faktura Set<Polozka>



- “Owning entity” - vlastník vztahu

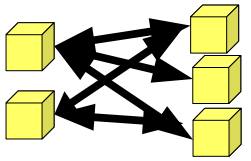
```
@Entity public class Auction {
  @Id long id;
  @OneToMany( mappedBy="auction", cascade=CascadeType.ALL)
  Set<Bid> bids = new HashSet();
  // ... + addBid(), getHighestBid(), ..
}
```

- V druhé entitě namapujeme N:1

```
@Entity public class Bid {
   @Id long id;
   @ManyToOne
   @JoinColumn(name="id_auction", nullable=false)
   Auction auction;
   ...
}
```

Viz http://download.oracle.com/javaee/6/api/javax/persistence/OneToMany.html

Faktura Set<Polozka>



JPA – obousměrná relace N:M

- Nejčastější řešení: Rozdělení na dva vztahy 1:N s “mezientitou”
- Jinak lze použít **`@ManyToMany`**
- Pozor při použití **`FetchType.EAGER`**
  - kartézský součin – může načíst obrovské počty entit!

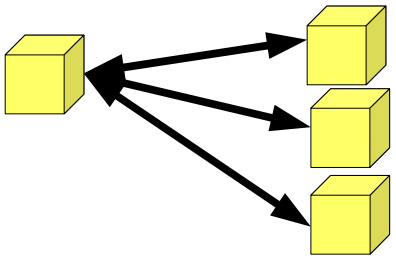
```
// In class User:
@ManyToMany
@JoinTable(name="user_fav_auction",
  joinColumns=
    @JoinColumn(name="id_user", referencedColumnName="id"),
  inverseJoinColumns=
    @JoinColumn(name="id_auction", referencedColumnName="id")
)
public Set<Auction> getFavoriteAuctions() { return favAuct; }
```

```
// in class Auction:
@ManyToMany(mappedBy="favoriteAuctions")
public Set<User> getFavoritedBy() { return favoritedBy; }
```

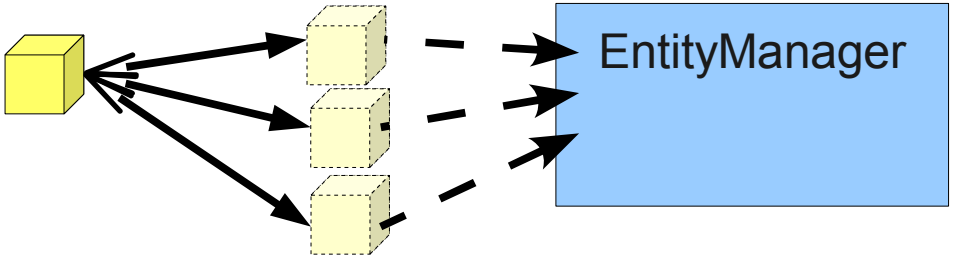
```
CREATE TABLE user_fav_auction ( id_user INT FK, id_auction INT FK )
```

JPA – FetchType.LAZY vs. EAGER

- EAGER



- LAZY

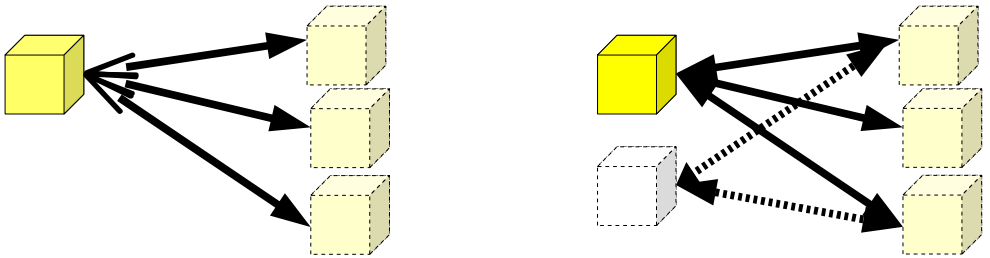


- ***LAZY*** vytvoří “proxy” objekt, který obsahuje jen ID entity
- Při prvním přístupu k entitě je automaticky načtena z DB.

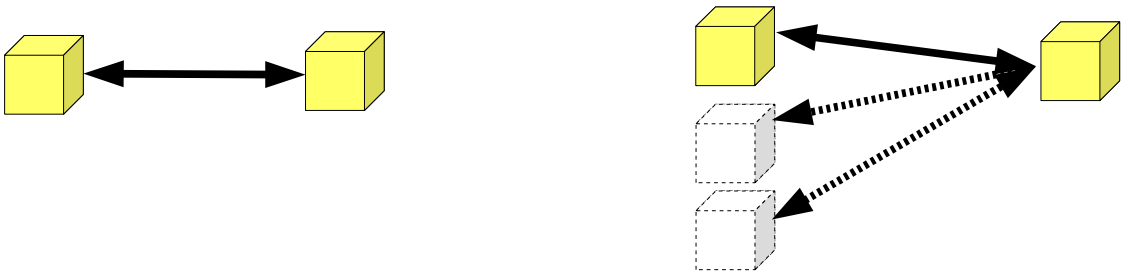
```
@Entity public class Auction {
    @Id long id;
    @OneToMany( fetch=FetchType.LAZY )
    @JoinColumn( name="id_auction", nullable=false )
    Set<Bid> bids;
    ...
}
```

JPA – FetchType.LAZY vs. EAGER

- **`@OneToMany`**, **`@ManyToMany`**: default je **`LAZY`**



- **`@OneToOne`**, **`@ManyToOne`**: Default je **`EAGER`**

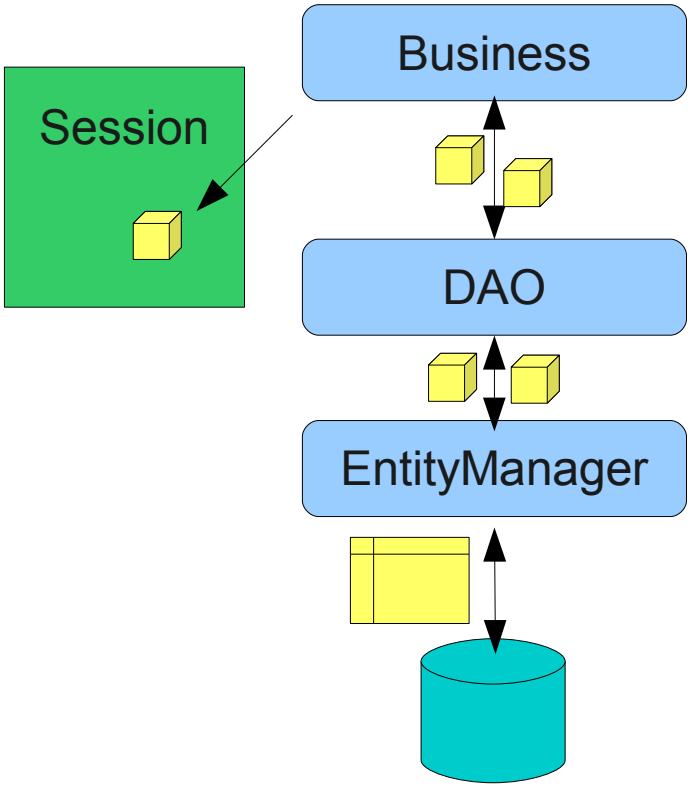


Viz http://download.oracle.com/javaee/6/api/javax/persistence/FetchType.html

JPA – attached, detached

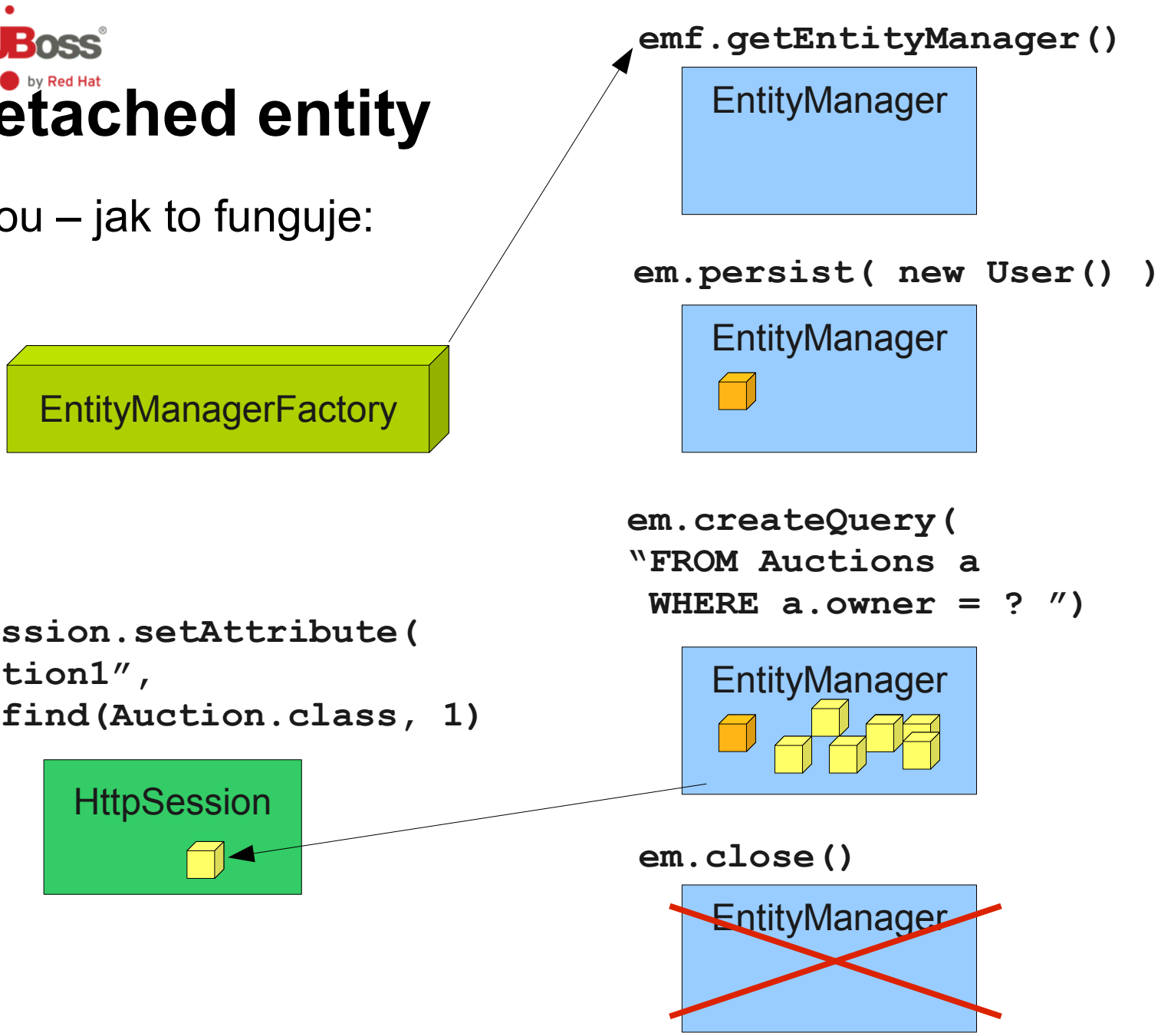
- Mezi akcemi uživatele nejsou entity potřeba
- → jsou uvolněny a mohou do gc()

- Ale co když si entitu někam uložíme?
  - např. do session
- Entita se po zavření session dostane do stavu *“detached”*
  - EntityManager ji nesleduje
- Pro opětovné připojení:
  - `em.merge(entity);`



JPA – detached entity

- Ještě jednou – jak to funguje:

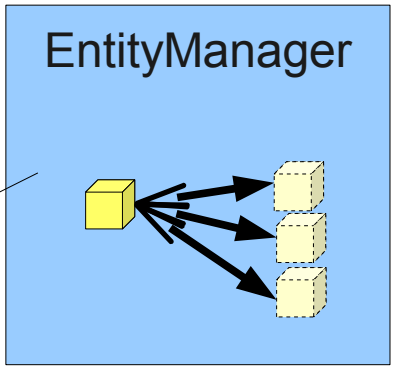


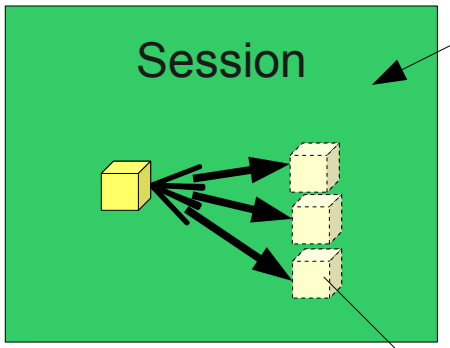
Aukce zůstává v session;
Z pohledu JPA je ve stavu *detached*.

LazyInitializationException

```
emf.getEntityManager()
```


```
em.createQuery(
  "FROM Auctions a
   WHERE a.id = ? ")
```






```
em.close()
```

???

JP-QL – Dotazy do databáze – definice

- JPA Query Language – JP-QL

```
@NamedQuery( name="findAuctionWinner",
    query="SELECT a.bids.user
           FROM Auction a
           WHERE a = :auction
           ORDER BY a.bids.amount DESC"
 )
public class Auction {
  ...
}
```

Viz http://docs.jboss.org/hibernate/entitymanager/3.6/reference/en/html/queryhql.html
a http://docs.jboss.org/hibernate/core/3.6/reference/en-US/html/queryhql.html

JP-QL – Dotazy do databáze – použití

```
public class AuctionDAO {
  public User findWinnerFor( Auction au ){
    Query query = em.createNamedQuery("findAuctionWinner");
    query.setParameter("auction", au);
    return (User) query.getSingleResult();
  }
}
```

SQL – Nativní dotazy do databáze

- Volání procedur, proprietární SQL (vendor lock-in)

```
Query q = em.createNativeQuery(
                  "CALL purgeOldAuctions()" );

TypedQuery<Auction> q =
        em.createNativeQuery(
          "SELECT 121, 115, 'Rovnač na ohybač'",
          Auction.class );
Auction a = q.getSingleResult();
```

Viz http://docs.jboss.org/hibernate/core/3.5/reference/en/html/queryhql.html

JPA – equals(Object), hashCode()

- Více java instancí pro stejný záznam v DB (business objekt)
  - BO – objekt reálného světa: např. člověk, auto, bank. účet
- **Cíl: zajistit, aby EntityManager rozeznal tentýž objekt (entitu).**

- Co je “stejný”?
  - PK z databáze?
    - **`id_user INT UNSIGNED`**
  - Reálný identifikátor BO? - tzv. *“business key”:*
    - 860217/0765, CBP-12-59, 2157339001/5500
  - Vlastnosti?
    - Jméno + rodné příjmení + místo narození + ...

- Upřednostňujte *business key*.
- Jinak podle situace.

Viz http://docs.jboss.org/hibernate/core/3.5/reference/en/html/persistent-classes.html#persistent-classes-equalshashcode .

JPA – OutOfMemoryException?

- Všude **`FetchType.EAGER`** ?

- Příliš mnoho dotazem nalezených položek
  - `SELECT Bid bid FROM Bid`

```
Query query = em.createQuery("SELECT p FROM Product p
        ORDER BY p.param1 ascending");
query.setFirstResult(100);
query.setMaxResults(20);
```

- Příliš mnoho operací v jediné session
  - `for ( int i=0; i<100000; i++ ) { em.persist( new User() ); }`

- Viz **`em.clear()`** a **`em.evict()`** - jen pokud víte, co děláte.

- http://docs.jboss.org/hibernate/core/3.5/reference/en/html/batch.html

JPA versus Hibernate - názvosloví

```
// JPA:
EntityManager em = emFactory.getEntityManager();
EntityTransaction tx = em.getTransaction();
tx.begin();
em.persist( new Auction( ... ) );
tx.commit();
em.close();
```

```
// Hibernate:
Session session = sessionFactory.openSession();
Transaction tx = session.beginTransaction();
session.save( new Auction( ... ));
tx.commit();
session.close();
```

JPA – Transakce (TBD - naťuknutí)

- O transakce se stará TransactionManager.
- EntityManager získává referenci na TM při vzniku
- EntityManager jen předává transakci, její řízení neprovádí.

```
EntityManager em = emFactory.getEntityManager();
EntityTransaction tx = em.getTransaction();
tx.begin();
em.persist( new Auction( ... ) );
tx.commit();
em.close();
```

ozizka@redhat.com | www.jboss.org