JPA támogatás Eclipse-ben

Tartalom

- ORM alapok
- JPA technológia
- JPA használatát segítő Eclipse technológiák

ORM alapok

Object-Relational Mapping

Object-Relational Mapping

- Cél: objektumok tárolása adatbázisban
- Probléma: az objektum-orientált és a relációs adatmodell közötti különbségek
- Megoldás: automatizált leképezés

Object-Relational Mapping

- Leképezés:

Osztályhierarchia	Séma
Osztály	Tábla
Mező	Oszlop
Objektum	Sor

Object-Relational Mapping

- Példa:

```
class Book {
      int id;
      String title;
      String author;
}
```

```
CREATE TABLE Book (
      INTEGER id,
      VARCHAR(255) title,
      VARCHAR(255) author
);
```

Problémás esetek

- Hivatkozáshoz objektumoknál van referencia, ehelyett kell elsődleges kulcs
- Öröklés
  - Többféle stratégia, ld. később
- Eltérő adattípusok
- Mikor szinkronizáljunk a memóriában levő modell és az adatbázis között?

ORM keretrendszerek

- Sok különböző
  - ActiveObjects
    - Öröklés + annotációk
  - Torque
    - Kódgenerálás XML leíró fájlokból
  - **JPA**
    - Annotációk és/vagy XML

JPA technológia

Java Persistence API

Java Persistence API

- EJB 3 specifikáció része
- Elfedi az adatbázisszerver-specifikus részleteket
- Futásidőben átlátszóan setterekkel tudjuk módosítani az objektumokat

JPA providerek

- A JPA csak egy API specifikáció
- Megvalósítását providerek biztosítják
  - Hibernate
  - OpenJPA
  - Toplink
  - **EclipseLink**

A JPA használata

- Java osztályok (POJO) annotációkkal
  - Alternatív módon: leképezés adatok XML-ben
    - Felülírja az annotációkat
    - Nem használjuk
- Alapelem: Entity = perzisztens osztály
- Perzisztens modul minden olyan jar, amelynek META-INF könyvtárában van `persistence.xml` fájl, amiben pl. a provider kerül definiálásra
- `javax.persistence` csomag tartalmazza

Entitás megadása

- Java osztály `@Entity`-vel (`javax.persistence.Entity`) annotálva, default konstruktorral
- Általában szerializálható (`implements Serializable`)
- Kötelező elsődleges kulcs attribútum: `@Id`
  - többféle automatikus ID-generáló stratégia megadható a `strategy` paraméterben

Entitás attribútumai

- A perzisztens attribútumokat a kliensek getterek/setterek formájában érhetik el (JavaBean konvenció)
- Nem perzisztens (tranziens) attribútum: @Transient
- Attribútum típusa lehet:
  - primitív típus
  - `String, BigInteger, BigDecimal, java.util.Date, java.util.Calendar, java.sql.Date, java.sql.Time, java.sql.Timestamp, byte[], Byte[], char[], Character[]`
  - `Enum`
  - más entitás, más entitások gyűjteménye
  - beágyazott osztály

Beágyazott osztály

- Olyan osztály, ami önmagában nem él perzisztens entitásként, csak egy perzisztens entitás példányához kapcsolódva, pl.

```
@Embeddable
public class EmploymentPeriod {
   java.util.Date startDate;
   java.util.Date endDate;
...
}
```

- Felhasználása egy entitáson belül:

```
@Embedded
@AttributeOverrides({
   @AttributeOverride(name="startDate", column=@Column("EMP_START")),
   @AttributeOverride(name="endDate", column=@Column("EMP_END"))
})
public EmploymentPeriod getEmploymentPeriod() { ... }
```

A leképezés testreszabása

- Az entitás, illetve az attribútumok nevével azonos a default tábla- és oszlopnév, ez felülírható
- `@Table(name="MyTable")`
  - `@SecondaryTable(s)` annotációval több táblába is szétszedhető
- `@Column(name="MyColumn")`
- Az oszlopoknak egyéb paraméterei is vannak
  - `nullable`
  - `unique`
  - `length`

Öröklés

- Fontos OO koncepció, melynek leképezése relációs adatbázisra nem triviális
- EJB3-tól kezdve
- Lehetséges leképezési módok:
  - egy tábla egy osztályhierarchiához
  - külön tábla gyermekosztályonként, hivatkozással
  - egy tábla egy konkrét osztályhoz

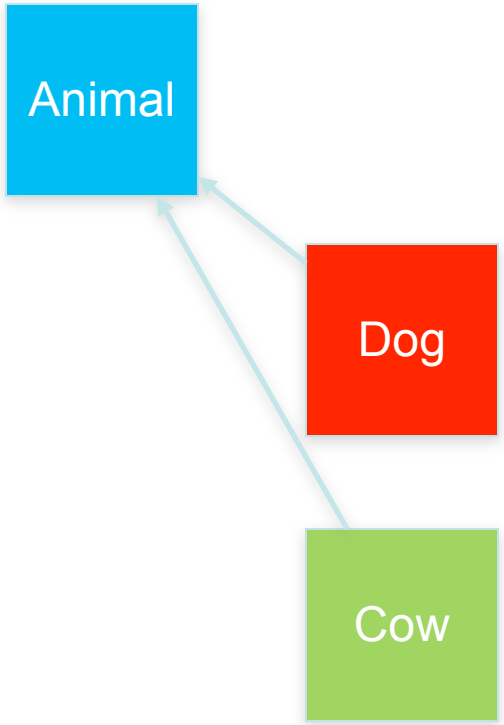
Egy tábla egy osztályhierarchiához

- egy táblában minden gyermekosztály
- diszkriminátor oszlop írja le a típust
- + hatékony (nem kell join)
- + polimorfizmust támogatja
- - mély hierarchia eseten sok, fölösleges oszlop
- - a gyermekosztályok attribútumainak megfelelő oszlopoknak nullázhatóknak kell lenniük

Egy tábla egy osztályhierarchiához

- A hierarchia legfelső szintjén:
  - @Inheritance(strategy=InheritanceType.SINGLE_TABLE)
  - @DiscriminatorColumn(name=<*oszlopnév*>)
- A hierarchia összes osztályánál:
  - @DiscriminatorValue(<*típusra utaló érték*>)

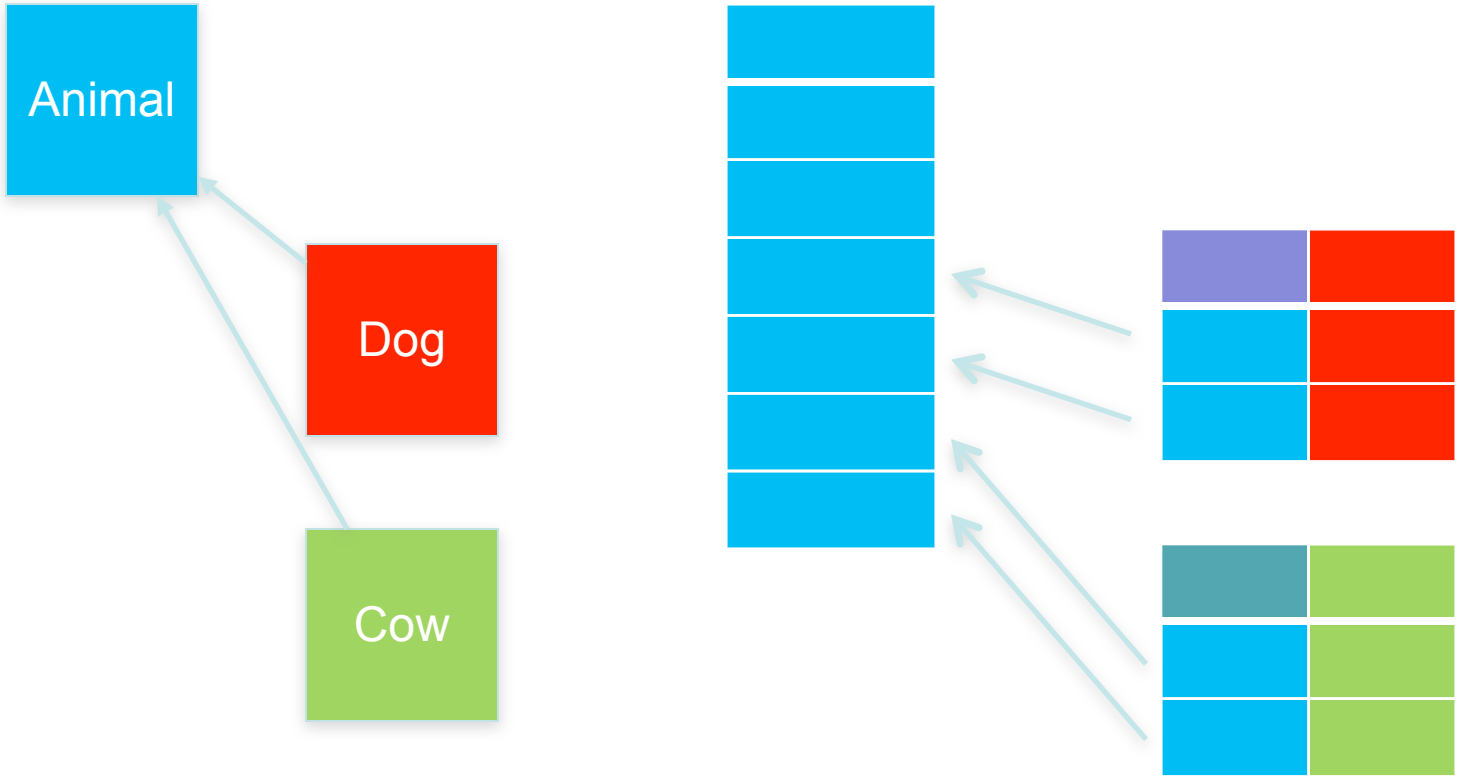
Egy tábla egy osztályhierarchiához



Külön tábla gyermekosztályonként

- Az ősosztályban definiált oszlopok egy táblában
- A gyermekosztályokban definiált oszlopok külön táblákban + idegen kulcs az ősre
- + nincsenek fölösleges oszlopok
- + definiálható nem nullázható oszlop
- + polimorfizmust támogatja
- - mély hierarchia esetén a sok join rontja a teljesitményt
- @Inheritance(strategy=InheritanceType.JOINED)

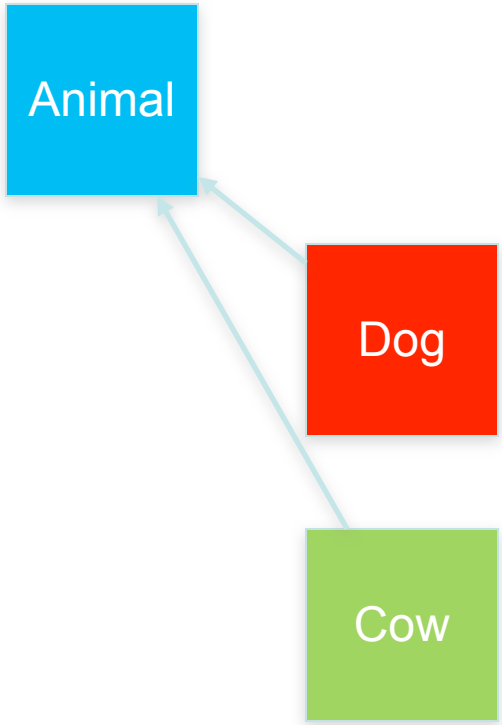
Külön tábla gyermekosztályonként



Önálló táblák gyermekosztályonként

- Külön tábla minden altípushoz
- Minden tábla tartalmazza az ősosztály attribútumait is
- \+ hatékony
- \- polimorfizmus támogatása nehézkes (oszlopdefiníciók többször szerepelnek)
- az EJB 3 nem követeli meg a támogatását

Önálló táblák gyermekosztályonként



Egyéb öröklési lehetőségek

- Entitás származhat nem entitásból
  - @MappedSuperClass-szal megjelölve az ősosztályt, annak nem lesz külön táblája egyik leképezési stratégia esetén sem, de az ottani attribútumok adatbázisba kerülnek
  - nem megjelölve egyáltalán nem kerülnek bele
- Nem entitás származhat entitásból
- Entitás lehet absztrakt
  - nem példányosodhat, de le lehet képezni táblába
  - le lehet kérdezni

Relációk

- Számosság szerint négyféle:
  - @OneToOne
  - @OneToMany
  - @ManyToOne
  - @ManyToMany
- Irány szerint kétféle:
  - Egyirányú
  - Kétirányú (a kapcsolat mindkét végén levő entitásnak lesz kapcsolatmenedzselő getter/setter metódusa): mappedBy paraméter
- Kétirányú OneToMany = Kétirányú ManyToOne
- A kapcsolatnak mindig egy tulajdonos oldala van

Példa reláció definiálására

- Tulajdonos oldalon (Employee):

```
@ManyToOne
@JoinColumn(name="company_id")
private Company company;
```

- Másik oldalon (Company):

```
@OneToMany(mappedBy="company_id")
private Collection<Employee> employees;
```

- + getterek, setterek
- a @JoinColumn helyett @JoinTable használandó, ha külön tábla tartja nyilván a kapcsolatot (pl. ManyToMany esetén mindenképpen)
- A @ManyToOne kötelezően tulajdonos oldal, mert nincs `mappedBy` paramétere, többi esetben szabadon választható a tulajdonos oldal

Relációk kaszkádosítása

- Mind a 4 kapcsolatdefiniáló annotációhoz megadható egy cascade elem, pl. `@OneToMany(cascade={ CascadeType.PERSIST, CascadeType.MERGE})`
- Lehetséges értékek:
  - PERSIST
  - MERGE
  - REMOVE
  - REFRESH
  - ALL
- Azt adja meg, milyen műveletek hívódjanak meg a kapcsolódó entitásokra is
- Default: nincs cascade

Fetch

- Mind a 4 kapcsolatdefiniáló annotációhoz megadható egy cascade elem, pl. `@OneToMany(fetch=FetchType.LAZY)`
- Azt adja meg, hogy egy entitás betöltésekor betöltődjenek-e a kapcsolódó entitások is
- **LAZY** (lusta): nem töltődnek be, csak ha hivatkozunk rájuk
  - így nem foglal memóriát, csak ha szükség van rá, de +1 lekérdezés
- **EAGER** (mohó, ez a default): kezdetben betöltődnek
  - gyorsabb, de több memóriát foglal
- Finomhangolási lehetőség:
  - legyen LAZY, de azokban a lekérdezésekben, ahol tudjuk, hogy szükség lesz a kapcsolódó entitásokra, használjunk fetch join-t az EJB-QL queryben, pl.

```
SELECT c form Customer c LEFT JOIN FETCH c.orders
```

Lazy fetch problémák

- Lecsatolt állapotban csak a korábban már hivatkozott kapcsolódó objektumokat fogja tartalmazni az entitás
- Ha egy kapcsolatait a lecsatolódás miatt részben elveszítő entitást vissza-merge-elünk, az adatbázisban is törlődnek a kapcsolatok
- A Lazy csak tanács, a persistence provider dönthet úgy, hogy mégis betölti mohó módon a kapcsolatokat

Perzisztencia kontextus

- A persistence provider által kezelt entitások egy halmaza
- Azonosítás a persistence unit nevével
- Entity manager lekérése pl.:

```
EntityManagerFactory factory =
  Persistence.createEntityManagerFactory(
  PERSISTENCE_UNIT_NAME);
EntityManager entityManager =
  factory.createEntityManager();
```

Entity Manager

- Entitások kezeléséért felelős
- Használata:
  - Entitások életciklusának kezelése
  - Szinkronizáció az adatbázissal
  - Entitások keresése lekérdezésekkel

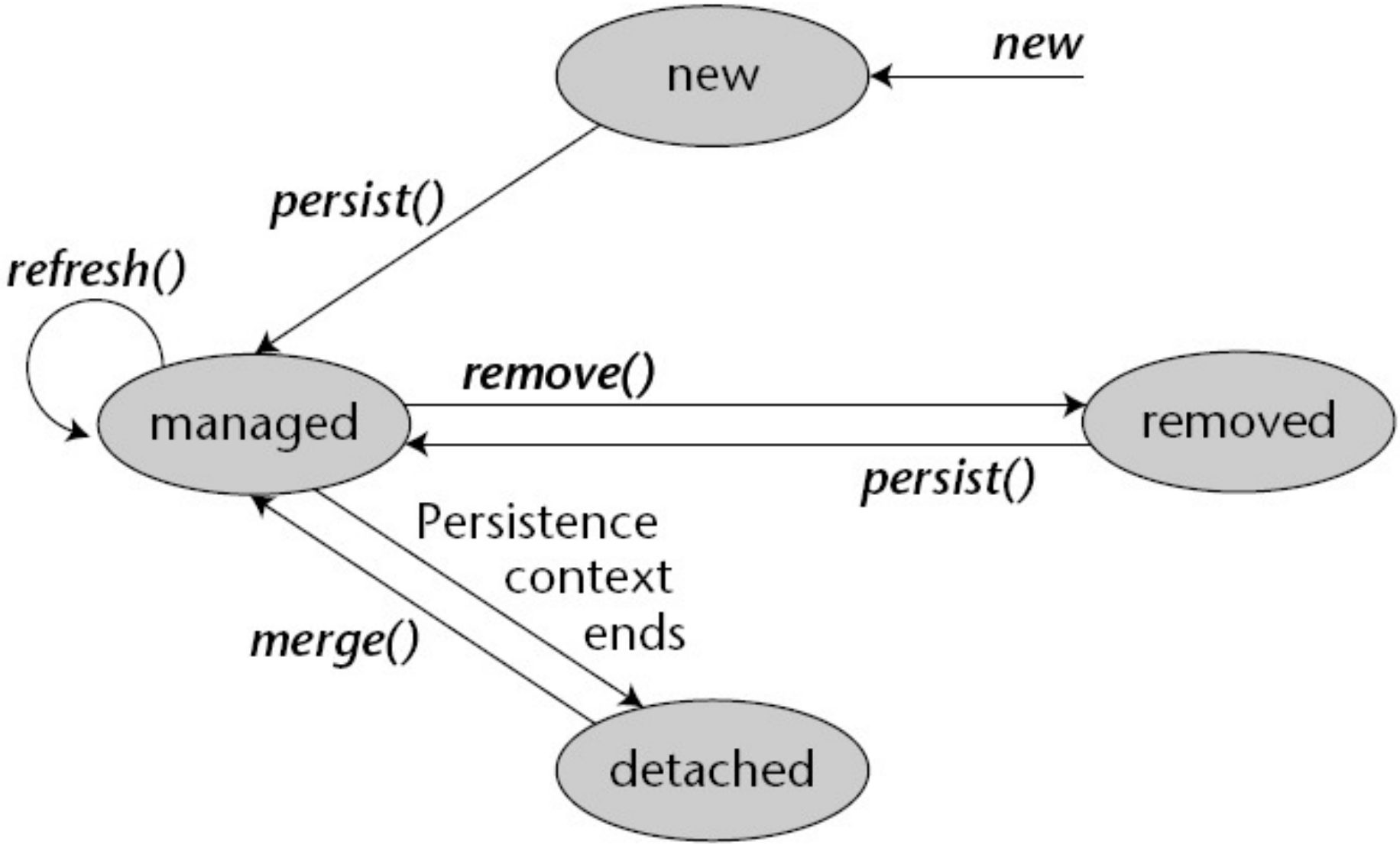
Tranzakciókezelés

- Tulajdonságok:
  - **A**tomic
  - **C**onsistent
  - **I**solated
  - **D**urable
- Hívások:
  - `entityManager.getTransaction().`
    - `begin()`
    - `commit()`
    - `rollback()`

Entitások állapotai

- **new**: new-val létrehozva kerül ide, csak a memóriában létezik, a módosítások nem mennek adatbázisba
- **managed**: létezik az adatbázisban, és hozzátartozik egy perzisztencia kontextushoz. Ez azzal jár, hogy a módosítások tranzakció commit végén, vagy explicit flush() hívással bekerülnek az adatbázisba
- **detached**: adatbázisban megvan, de nem tartozik perzisztens kontextushoz; ebben az állapotban olyan, mint egy DTO (Data Transfer Object)
- **removed**: még perzisztencia kontextushoz tartozik, de már ki van jelölve, hogy törölve lesz az adatbázisbol

Entitások életciklusa



Entitás életciklus callbackek

- Metódusok a következő annotációk valamelyikével:
  - @PrePersist
  - @PostPersist
  - @PreRemove
  - @PostRemove
  - @PreUpdate
  - @PostUpdate
  - @PostLoad
- A persistence provider hívja őket
- Ha külön osztályba akarjuk rakni, @EntityListeners-ben kötjük hozzá az entitáshoz az osztályt, és a metódusok az entitást megkapják paraméterül

Adatbázis szinkronizáció

- Általában commitkor automatikusan megtörténik
- Explicit módon is megtehetjük az EntityManageren keresztül:
  - `flush(entity)`: beírja a változtatásokat
  - `refresh(entity)`: beolvassa a változtatásokat

Lekérdezések

- Egyszerű keresés elsődleges kulcs alapján:

```
<T> T find(Class<T> entityClass, Object primaryKey)
```

- Bonyolultabb lekérdezések:
  - EJB-QL nyelven: `public Query createQuery(String ejbqlString)`
  - natív SQL: `public Query createNativeQuery(String sqlString)`

Lekérdezések

- Paraméterkezelés (biztonság):
  - `setParameter(String, Object)/setParameter(int, Object)`: név vagy index alapján
- Eredmény lekérése:
  - `getSingleResult()`
  - `getResultList()`
- Módosítás/törlés
  - `executeUpdate()`
  - Lehetőség van tömeges törlésre/módosításra

Konkurenciakezelés

- Két lehetőség
  - Optimista
    - @Version-nel meg kell jelölni egy egész szám vagy TimeStamp típusú attribútumot
    - ezt a persistence provider kezeli (update-kor növeli), kódból nem módosítjuk
    - ha beíráskor azt látja, hogy a verziószám módosult (egy konkurens kliens módosítása miatt), nem módosít, hanem OptimisticLockException-t dob
  - Explicit zárak
    - `entityManager.lock(Object entity, LockMode)`
    - LockMode: READ vagy WRITE lehet
    - csak tranzakción belül hívható

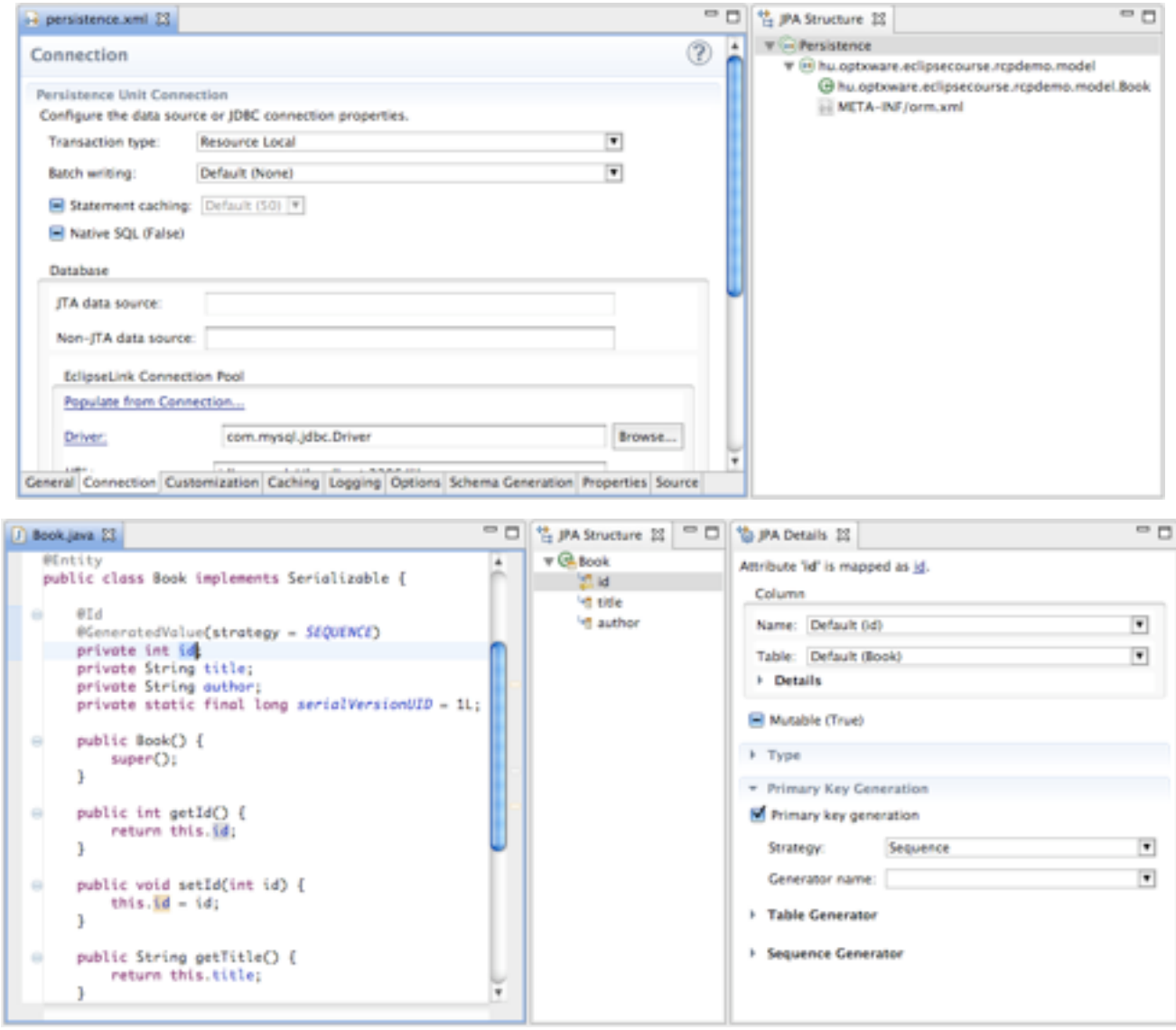
JPA használatát segítő Eclipse technológiák

Eclipse Dali, EclipseLink

Eclipse Dali

- A Web Tools Platform része
  - Természetesen Java SE-vel is használható
- Forms alapú szerkesztőfelület
  - Persistence unit
  - Entity
- Függőségkezelés megkönnyítése
- Integráció bármelyik providerrel, de EclipseLinkkel a legkényelmesebb
- Egyéb eszközök:
  - Entitások és táblák közötti konverzió
  - Annotált & persistence.xml-ben felsorolt entitások szinkronizációja

Eclipse Dali



EclipseLink

- TopLinkre épül
- Jól használható OSGi környezetben
- Együttműködik a Dalival
- `org.eclipse.persistence.jpa` bundle

EclipseLink-specifikus propertyk

- A provider neve
  `org.eclipse.persistence.jpa.PersistenceProvider`
  - OSGI környezetben
    `org.eclipse.persistence.jpa.osgi.PersistenceProvider`
- Automatikus sémalétrehozás
  - `<property name="eclipselink.ddl-generation" value="drop-and-create-tables"/>`
    - vagy `create-tables`
  - `<property name="eclipselink.ddl-generation.output-mode" value="database"/>`

Használat

- EntityManager lekérése

```
EntityManager entityManager =
new org.eclipse.persistence.jpa.osgi.PersistenceProvider().
createEntityManagerFactory(PERSISTENCE_UNIT_NAME).
createEntityManager();
```

- Tranzakció kezdés
  - `entityManager.getTransaction().begin();`
- Tranzakció befejezés
  - `entityManager.getTransaction().commit();`
- EntityManager lezárása
  - `entityManager.close();`

További források

- Prezentációk
  - http://www.slideshare.net/junyuo/java-persistence-api-jpa-step-by-step-presentation
  - http://www.slideshare.net/caroljmcdonald/persistencecmcdonaldmainejug3
- Tutorialok
  - http://www.vogella.de/articles/JavaPersistenceAPI/article.html
  - http://www.vogella.de/articles/EclipseDali/article.html