



Platforma Java

Objektově relační mapování II

Petr Krajča



Katedra informatiky
Univerzita Palackého v Olomouci



- vyhledání objektu podle klíče (resp. průchod grafem objektů)
- ad-hoc dotazy (JPQL)
- pojmenované dotazy (JPQL)
- Criteria API ()
- nativní dotazy (SQL)

Vyhledání podle klíče

```
em.find(StockItem.class, value);
```

Ad-hoc dotazy

```
Query q = em.createQuery("select item from Invoice item");  
List<Invoice> invoices = q.getResultList();
```

Ad-hoc dotazy s parametry

```
Query q = em.createQuery("select cust from Customer cust where"  
                        + " cust.itin = :itin");  
q.setParameter("itin", value);  
return q.getSingleResult();
```

Nikdy nelepte parametry do dotazu pomocí spojení řetězců!



- obdoba prepared statements

```
@NamedQuery(  
    name = "allInvoicesForCustomer",  
    query = "select i from Invoice i where i.customer = :cust")  
public class Invoice implements Serializable { /* ... */ }
```

Vyvolání dotazu

```
Query q = em.createNamedQuery("allInvoicesForCustomer");  
q.setParameter("cust", customer);  
List<Invoice> invoices = q.getResultList();
```



Více pojmenovaných dotazů pro jednu entitu

```
@NamedQueries({
    @NamedQuery(
        name = "allInvoices",
        query = "select i from Invoice i"),
    @NamedQuery(
        name = "allInvoicesForCustomer",
        query = "select i from Invoice i where i.customer = :cust"),
    @NamedQuery(
        name = "allInvoicesForCustomerItin",
        query = "select i from Invoice i where i.customer.itin = :custItin")
})
@Entity
public class Invoice implements Serializable { /* ... */ }
```



- vhodné pro hromadné aktualizace

```
Query q = em.createQuery("UPDATE StockItem it SET"  
    + " it.unitPrice = it.unitPrice * 1.1");  
int updated = q.executeUpdate();
```

```
Query q = em.createQuery("DELETE FROM StockItem it"  
    + " WHERE it.itemName = :name");  
q.setParameter("name", ...);  
int updated = q.executeUpdate();
```



- rozsah operací podobný jako v SQL
- možné vrátit obecný tuple jako Object []

```
List<Object[]> res = em.createQuery("SELECT c.name, COUNT(inv.id)"  
    + " FROM Customer c, Invoice inv"  
    + " WHERE inv.customer = c "  
    + " GROUP BY c.name").getResultList();
```



- typově bezpečné
- podpora dynamické tvorby dotazů
- meta-model – vygenerované třídy s příponou `_`, např. `Foo_` pro entitu `Foo`
- **CriteriaBuilder** – třída sestavující dotaz
- **CriteriaQuery** – třída představující dotaz (je potřeba převést na běžný dotaz `EntityManageru`)

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery cq = cb.createQuery(Invoice.class);
Root<Invoice> invoice = cq.from(Invoice.class);
cq.select(invoice);
```

```
TypedQuery<Invoice> q = em.createQuery(cq);
List<Invoice> invoices = q.getResultList();
```



```
Root<Invoice> invoice = cq.from(Invoice.class);  
cq.where(cb.equal(invoice.get(Invoice_.customer), customer));  
cq.select(invoice);
```

- `cb.equal` – operace porovnání
- `Invoice_.customer` – odkaz na atribut v meta-modelu

Aktualizace

```
CriteriaBuilder update = entityManager.getCriteriaBuilder();  
CriteriaUpdate<StockItem> cq = update.createCriteriaUpdate(StockItem.class);  
Root<StockItem> item = cq.from(StockItem.class);
```

```
cq.set(StockItem_.unitPrice,  
    update.prod(item.get(StockItem_.unitPrice), new BigDecimal(2)));
```

```
Query q = entityManager.createQuery(cq);  
q.executeUpdate();
```



- anotace @Inheritance
- nesoulad rel. modelu a objekt. modelu
- tři možnosti

SINGLE_TABLE

- jedna tabulka pro celou hierarchii
- jeden sloupec pro rozlišení jednotlivých tříd
- atributy, které nejsou v dané třídě definovány nastaveny na NULL
- potenciální problémy
 - chybějící sloupec určující třídy
 - atributům v podtřídě nelze nastavit NOT NULL

```
@Entity
@Inheritance
@DiscriminatorColumn(name="INVOICE_TYPE")
@ForceDiscriminator
public abstract class Invoice {
    @Id
    private long id;
    /* ... */
}
@Entity
@DiscriminatorValue("NAT")
public class NationalInvoice extends Invoice {
    private String eetNo;
}
@Entity
@DiscriminatorValue("INT")
public class InternationalInvoice extends Invoice {
    private String iban;
}
```



JOINED

- každá entita má tabulku s atributy, které jsou v ní definované
- potomci a jejich tabulky musí obsahovat atribut—primární klíč—který je definovaný v kořenové třídě
- přirozený způsob reprezentace
- některé implementace JPA (ne)vyžadují sloupec určující typ třídy

```
@Entity
@Inheritance(strategy=InheritanceType.JOINED)
@DiscriminatorColumn(name="INVOICE_TYPE")
@ForceDiscriminator
public abstract class Invoice {
    @Id
    private long id;
    /* ... */
}

@Entity
@DiscriminatorValue("NAT")
@Table(name = "INVOICE_NAT")
public class NationalInvoice extends Invoice {
    private String eetNo;
}

@Entity
@DiscriminatorValue("INT")
@Table(name = "INVOICE_INT")
public class InternationalInvoice extends Invoice {
    private String iban;
}
```



TABLE_PER_CLASS

- pro každou třídu existuje jedna tabulka
- každá tabulka obsahuje všechny atributy včetně atributů předka
- pomalé dotazy (vícenásobné dotazy nebo UNIONy)
- problematická spojení a řazení záznamů

@MappedSuperclass

- podobný přístup jako TABLE_PER_CLASS
- nevyskytuje se v datovém modelu, nelze ji odkazovat
- každý potomek má vlastní tabulku



```
@Entity
@Inheritance(strategy=InheritanceType.TABLE_PER_CLASS)
public abstract class Invoice {
    @Id
    private long id;
    /* ... */
}

@Entity
@Table(name = "INVOICE_NAT")
public class NationalInvoice extends Invoice {
    private String eetNo;
}

@Entity
@Table(name = "INVOICE_INT")
public class InternationalInvoice extends Invoice {
    private String iban;
}
```