Agenda

Query

JPAQL

Criteria API

SQL

Batch-Updates

Performance

Cache

Criteria API

- Abfragen objektorientiert bauen
- Typsicherheit, da keine String-Verkettungen

```
Query q = em.createQuery("SELECT u FROM User u
   where u.username =  :username")
   .setParameter("username",   ...).getResultList();

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<User> c = cb.createQuery(User.class);

Root<User> userRoot = c.from(User.class);

c.select(userRoot)

c.where(cb.equal(userRoot.get("username"), username));

List<User> result = em.createQuery(c).getResultList();
```

Criteria Schlüsselworte

Schlüsselwort	Criteria API Interface	Methode
SELECT	CriteriaQuery	select()
	Subquery	select()
FROM	AbstractQuery	from()
WHERE	AbstractQuery	where()
ORDER BY	CriteriaQuery	orderBy()
GROUP BY	AbstractQuery	groupBy()
HAVING	AbstractQuery	having()

From-Klausel

- **from() erzeugt Query-Root-Objekt**

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<User> c = cb.createQuery(User.class);
Root<User> userRoot = c.from(User.class);
```

- **mehrere Entitäten in from-Klausel (Join)**

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<User> c = cb.createQuery(User.class);
Root<User> userRoot = c.from(User.class);
Root<Customer> customerRoot = c.from(Customer.class);
```

Select-Klausel

- **Entität selektieren**

```
c.select(userRoot);
```

- **Felder selektieren**

```
c.select(userRoot.get("username"));
c.select(cb.tuple(userRoot.get("username"),
userRoot.get("email")));
c.multiselect(userRoot.get("username“).alias("user") ,
  userRoot.get("email")) ;
```

- **Distinct**

```
c.select(userRoot).distinct(true);
```

Where - CriteriaBuilder-Methoden

- and(), or(), not()
- Equals(), notEqual(), greaterThan(), gt(), greaterThanOrEqualTo(), ge(), lessThan(), lt(), lessThanOrEqualTo(), le(), between(), isNull(), isNotNull(), exists(), not(exists())
- isEmpty(), isNotEmpty(), isMember(), isNotMember(), like(), notLike(), in(), not(in())
- abs(), lower(), upper(), length(), trim(), ...
- avg(), sum(), min(), least(), max(), greatest(), count(), countDistinct()

Where - Predicates


```
Root<User> userRoot = c.from(User.class);
Predicate username =
cb.equal(userRoot.get("username"),  username);
Predicate mail =
cb.like(userRoot.get("email"),  email);
c.select(userRoot).where(cb.and(username,  mail));
```

Join

- **Standard JoinType.INNER**
- alternativ JoinType.LEFT oder JoinType.RIGHT
- JoinType.RIGHT muß nicht vom Provider unterstützt werden

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<User> c = cb.createQuery(User.class);
Root<User> userRoot = c.from(User.class);
userRoot.join("addresses",  JoinType.LEFT).alias("a");
```

- **Fetch Joins**

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<User> c = cb.createQuery(User.class);
Root<User> userRoot = c.from(User.class);
userRoot.fetch("addresses")
```

Sortieren

- über mehrere Spalten auf- und absteigend sortierbar

```
Root<User> userRoot = c.from(User.class);
c.select(userRoot).where(
  cb.equal(userRoot.get("username"),
           username));
c.orderBy(cb.asc( userRoot, get("email") ) );
```

Aggregationen


```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Object[]> c =
cb.createQuery (Object [].class ) ;
Root<Person> root = c.from(Person.class);
c.multiselect(
   cb.count(root) ,
   cb.avg(root.get(Person_.age))
   cb.max(root.get(Person_.age)));
c.groupBy(root.get(Person_.firstName));
c.having(cb.like(root.get(Person_.lastName), "Müller");
em.createQuery(c).getResultList();
```

Canonical Metamodel – Stark typisierte Abfragen

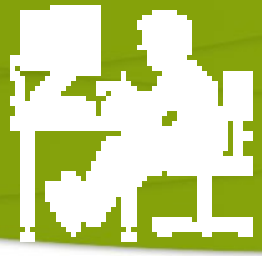
- de.example.User => de.example.User_
- User_:
  - javax.persistence.StaticMetamodelAnnotation
  - statische Felder

```
public static volatile SingularAttribute<User, String> username;
public static volatile CollectionAttribute<User, Address> addresses;

        …
c.where(
cb.equal(userRoot.get(User_.username),  username));
List<User> result = em.createQuery(c).getResultList();
```

- Hibernate JPA 2 Metamodel Generator

Aufgabe



- Aufgabe 12 (optional):
- Abfragen (Criteria API)

Queries by Criteria (Hibernate)

- Queries by Criteria (QBC) API
  - Queries werden objektorientiert zusammengebaut
  - Criteria Objekte werden zur Laufzeit manipuliert
  - Enthält Query by Example

```
Criteria criteria =
   session.createCriteria(StockDailyRecord.class);
List result = criteria.list() ;
```

QBC mit Parametern

- Der WHERE Clause wird durch Hinzufügen von Restrictions realisiert

```
Criteria criteria =
   session.createCriteria(StockDailyRecord.class)
   .add(Restrictions.eq("volume", 10000));
List result = criteria.list() ;
```

QBC Operatoren / Restrictions (WHERE)


```
session.createCriteria(Person.class)
      .add(Restrictions.between("age",new BigDecimal(1) ,
                        new BigDecimal(10))).list ();
session.createCriteria(Person.class)
   .add(Restrictions.gt("age", new
   BigDecimal(50))).list();

String[]  emails = {  "foo@example.de",
   "bar@example.de" };

session.createCriteria(Person,class)
   .add(Restrictions.in("email“,  emails)   )
   .list ();
```

QBC Operatoren / Restrictions (WHERE)


```
session.createCriteria(Person.class)
   .add(Restrictions.like("firstname",   "P",
   MatchMode.START))
   .list() ;
session.createCriteria(Person.class)
   .add(Restrictions.ilike("firstname",   "Peter",
       MatchMode.EXACT))
   .list();
session.createCriteria(Person.class)
   .add(Restrictions.ilike("firstname",   "ol",
       MatchMode.ANYWHERE) )
   .list();
```

QBC Und/Oder Verknüpfungen


```
Criterion alter1 = Restrictions.gt("age", 50);

Criterion alter2 = Restrictions.between("age",  20, 40);

Criterion ort = Restrictions.like("town", "Berlin");

session.createCriteria(Person.class)
   .add(Restrictions.or(alter1, alter2)
   .list ();

session.createCriteria(Person.class)
   .add(Restrictions.and(alter1, ort)
   .list();
```

QBC Sortieren

- Wird durch Klasse Order angegeben

```
Criteria criteria =
   session.createCriteria(StockDailyRecord.class)
   .addOrder( Order.asc("date") );
List result = criteria.list();
```

QBC Join: Mittels zusätzlicher Criteria


```
pcriteria = session.createCriteria(Person.class)
   .add(Restrictions.like("name", "Best");
acriteria = pcriteria.createCriteria("addresses") ;
acriteria.add( Restrictions.like("city",  "Mannheim"));
List results = pcriteria.list();
```

QBC Join: Mittels Alias


```
List results = session.createCriteria(Person.class)
   .createAlias("addresses", "adr")
   .add(Restrictions.like("name", "best") )
   .add(Restrictions.like("adr.city", "Berlin"))
   .list();
```

QBC Eager Fetching

- im Vergleich zu HQL wird die globale Eager-Fetching-Strategie (aus den Mapping-Dateien) nicht ignoriert
- FetchMode sollte explizit angegeben werden

```
session.createCriteria(Person.class)
   .setFetchMode("roles",  FetchMode.JOIN)
   .add(Restrictions.like("firstname", "ab",
       MatchMode.ANYWHERE) )
   .list() ;
session.createCriteria(Person.class)
   .setFetchMode("roles",  FetchMode.SELECT)
   .add(Restrictions.gt("age", new BigDecimal(50)))
   .list();
```

QBC Distinct

- Es gibt kein Schlüsselwort distinct
- nur nachträgliches Filtern des Ergebnisses

```
Criteria c = session.createCriteria(...) ;
   c.setResultTransformer(Criteria.DISTINCT_ROOT_ENTITY);
```

QBC Aggregation

- Aggregationen als Projektion angeben

```
Criteria crit = session.createCriteria(Person.class)
   .setProjection(Projections.rowCount())
   .add(Restrictions.eq("firstname", "Peter"))
Integer result =  (Integer)crit.uniqueResult();
```

QBC Aggregation


```
List results = session.createCriteria(Person.class)
  .setProjection(Projections.projectionList()
  .add(Projections.rowCount(),   "countByFirstname")
  .add(Projections.avg("age"), "avgAge")
  .add(Projections.max("age"),  "maxAge")
.add(Projections.groupProperty("firstname"),"firstname") )
.addOrder( Order.desc("countByfirstname") )
.addOrder( Order.desc("avgAge") )
.list();
```

Queries by Example (QBE)

- Teil von QBC
- Idee
  - Eine Instanz der abgefragten Klasse enthält Werte an einigen Attributen, die beim Suchen benutzt werden

```
Person examplePerson = new Person();
examplePerson.setFirstname("Peter");
Criteria criteria =
session.createCriteria(Person.class);
criteria.add( Example.create(examplePerson) );
List result = criteria.list();
```