IT-Academy

Hibernate
Criteria JPA

Автор: Юлий Слабко

Hibernate Criteria Queries

IT-Academy

- JPA Criteria – представляет объектно-ориентированную альтернативу JPQL.

```
@Test
public void getAllEmployee() {
    EntityManager em = HibernateUtil.getEntityManager();
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
    Root<Employee> employeeRoot = criteria.from(Employee.class);
    criteria.select(employeeRoot);
    List<Employee> employees = em.createQuery(criteria).getResultList();
    employees.forEach(System.out::println);
}
```

```
Employee{id=2, name='Yuli, age=27, salary=16000.99}
Employee{id=3, name='Max, age=38, salary=10000.0}
Employee{id=4, name='Paul, age=43, salary=15000.0}
Employee{id=5, name='Gleb, age=37, salary=15000.0}
Employee{id=6, name='Li, age=62, salary=13099.0}
Employee{id=7, name='Alex, age=22, salary=4500.0}
```

Hibernate Criteria Queries

IT-Academy

```
@Test
public void getAllEmployee() {
    EntityManager em = HibernateUtil.getEntityManager();
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
    criteria.from(Employee.class);
    List<Employee> employees = em.createQuery(criteria).getResultList();
    employees.forEach(System.out::println);
}
```

```
Employee{id=2, name='Yuli, age=27, salary=16000.99}
Employee{id=3, name='Max, age=38, salary=10000.0}
Employee{id=4, name='Paul, age=43, salary=15000.0}
Employee{id=5, name='Gleb, age=37, salary=15000.0}
Employee{id=6, name='Li, age=62, salary=13099.0}
Employee{id=7, name='Alex, age=22, salary=4500.0}
```

Restrictions with Criteria

IT-Academy

```
@Test
public void getEmployeeByNameTest() {
    EntityManager em = HibernateUtil.getEntityManager();
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
    Root<Employee> emp = criteria.from(Employee.class);
    criteria.select(emp)
            .where(cb.equal(emp.get("name"), "Yuli"));
    List<Employee> employees = em.createQuery(criteria).getResultList();
    employees.forEach(System.out::println);
}
```

select employee0_.id as id1_2_, employee0_.age as age2_2_, employee0_.DEPARTMENT_ID as DEPARTME5_2_, employee0_.name as name3_2_, employee0_.salary as salary4_2_ from Employee employee0_ **where** employee0_.**name=?**

Employee{id=2, name='Yuli, age=27, salary=16000.99}

Restrictions with Criteria (>)

IT-Academy

```
@Test
public void greaterTest() {
    EntityManager em = HibernateUtil.getEntityManager();
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
    Root<Employee> emp = criteria.from(Employee.class);
    criteria.select(emp)
            .where(cb.gt(emp.get("salary"), 10000));
    List<Employee> employees = em.createQuery(criteria).getResultList();
    employees.forEach(System.out::println);
}
```

```
Employee{id=2, name='Yuli, age=27, salary=16000.99}
Employee{id=4, name='Paul, age=43, salary=15000.0}
Employee{id=5, name='Gleb, age=37, salary=15000.0}
Employee{id=6, name='Li, age=62, salary=13099.0}
```

Restrictions with Criteria (<=)

IT-Academy

```
@Test
public void lessTest() {
    EntityManager em = HibernateUtil.getEntityManager();
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
    Root<Employee> emp = criteria.from(Employee.class);
    criteria.select(emp)
            .where(cb.le(emp.get("age"), 43));
    List<Employee> employees = em.createQuery(criteria).getResultList();
    employees.forEach(System.out::println);
}
```

select employee0_.id as id1_2_, employee0_.age as age2_2_, employee0_.DEPARTMENT_ID as DEPARTME5_2_, employee0_.name as name3_2_, employee0_.salary as salary4_2_
from Employee employee0_
where employee0_.**age<=43**

Employee{id=2, name='Yuli, age=27, salary=16000.99}
Employee{id=3, name='Max, age=38, salary=10000.0}
Employee{id=4, name='Paul, age=43, salary=15000.0}
Employee{id=5, name='Gleb, age=37, salary=15000.0}
Employee{id=7, name='Alex, age=22, salary=4500.0}

Restrictions with Criteria (like)

IT-Academy

```
@Test
public void likeTest() {
    EntityManager em = HibernateUtil.getEntityManager();
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
    Root<Employee> emp = criteria.from(Employee.class);
    criteria.select(emp)
            .where(cb.like(emp.get("name"), "%ul%"));
    List<Employee> employees = em.createQuery(criteria).getResultList();
    employees.forEach(System.out::println);
}
```

select employee0_.id as id1_2_, employee0_.age as age2_2_, employee0_.DEPARTMENT_ID as DEPARTME5_2_, employee0_.name as name3_2_, employee0_.salary as salary4_2_
from Employee employee0_
where employee0_.name **like** ?

Employee{id=2, name='Yuli, age=27, salary=16000.99}
Employee{id=4, name='Paul, age=43, salary=15000.0}

Restrictions with Criteria (between)

IT-Academy

```
@Test
public void betweenTest() {
    EntityManager em = HibernateUtil.getEntityManager();
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
    Root<Employee> emp = criteria.from(Employee.class);
    criteria.select(emp)
            .where(cb.between(emp.get("age"), 35, 50));
    List<Employee> employees = em.createQuery(criteria).getResultList();
    employees.forEach(System.out::println);
}
```

select employee0_.id as id1_2_, employee0_.age as age2_2_, employee0_.DEPARTMENT_ID as DEPARTME5_2_, employee0_.name as name3_2_, employee0_.salary as salary4_2_
from Employee employee0_
where employee0_.age between 20 and 35

Employee{id=3, name='Max, age=38, salary=10000.0}
Employee{id=4, name='Paul, age=43, salary=15000.0}
Employee{id=5, name='Gleb, age=37, salary=15000.0}

Restrictions with Criteria (isNotNull)

IT-Academy

```
@Test
    public void isNullTest() {
        EntityManager em = HibernateUtil.getEntityManager();
        CriteriaBuilder cb = em.getCriteriaBuilder();
        CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
        Root<Employee> emp = criteria.from(Employee.class);
        criteria.select(emp)
                .where(cb.isNotNull(emp.get("name")));
//              .where(cb.isNull(emp.get("name")));
        List<Employee> employees = em.createQuery(criteria).getResultList();
        employees.forEach(System.out::println);
    }
```

select employee0_.id as id1_2_, employee0_.age as age2_2_, employee0_.DEPARTMENT_ID as DEPARTME5_2_, employee0_.name as name3_2_, employee0_.salary as salary4_2_
from Employee employee0_
where employee0_.name **is not null**

```
Employee{id=2, name='Yuli, age=27, salary=16000.99}
Employee{id=3, name='Max, age=38, salary=10000.0}
Employee{id=4, name='Paul, age=43, salary=15000.0}
Employee{id=5, name='Gleb, age=37, salary=15000.0}
Employee{id=6, name='Li, age=62, salary=13099.0}
Employee{id=7, name='Alex, age=22, salary=4500.0}
```

LogicalExpression

```
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
Root<Employee> emp = criteria.from(Employee.class);
Predicate predicate = cb.and(
        cb.like(emp.get("name"), "%ul%"),
        cb.gt(emp.get("age"), 30)
);
criteria.select(emp).where(predicate);

List<Employee> employees = em.createQuery(criteria).getResultList();
employees.forEach(System.out::println);
```

select employee0_.id as id1_2_, employee0_.age as age2_2_, employee0_.DEPARTMENT_ID as DEPARTME5_2_, employee0_.name as name3_2_, employee0_.salary as salary4_2_
from Employee employee0_
where (employee0_.name like ?) **and** employee0_.age>30

Employee{id=4, name='Paul, age=43, salary=15000.0}

LogicalExpression

IT-Academy

```
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
Root<Employee> emp = criteria.from(Employee.class);
Expression<Integer> age = emp.get("age");
Predicate predicate = cb.and(
        cb.or(age.in(24, 28, 35), cb.equal(emp.get("name"), "Yulij")),
        cb.like(emp.get("name"), "%ul%"),
        cb.gt(emp.get("age"), 30)
);
criteria.select(emp).where(predicate);

List<Employee> employees = em.createQuery(criteria).getResultList();
employees.forEach(System.out::println);
```

select employee0_.id as id1_2_, employee0_.age as age2_2_, employee0_.DEPARTMENT_ID as DEPARTME5_2_,
employee0_.name as name3_2_, employee0_.salary as salary4_2_
from Employee employee0_
where
(employee0_.age **in** (24 , 28 , 35) **or** employee0_.name=?)
and (employee0_.name like ?)
and employee0_.age>30

Вопросы

IT-Academy

Pagination using Criteria

IT-Academy

Для постраничного вывода в hibernate существуют следующие методы:

- `TypedQuery<X> setFirstResult(int startPosition);`
- `TypedQuery<X> setMaxResults(int maxResult);`

```
EntityManager em = HibernateUtil.getEntityManager();
int pageNumber = 1;
int pageSize = 2;
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
criteria.from(Employee.class);
TypedQuery<Employee> typedQuery = em.createQuery(criteria);
typedQuery.setFirstResult(pageSize * (pageNumber-1));
typedQuery.setMaxResults(pageSize);
List<Employee> employees = typedQuery.getResultList();
```

select employee0_.id as id1_2_, employee0_.age as age2_2_, employee0_.DEPARTMENT_ID as DEPARTME5_2_, employee0_.name as name3_2_, employee0_.salary as salary4_2_ from Employee employee0_ limit ?

Employee{id=2, name='Yuli, age=27, salary=16000.99}
Employee{id=3, name='Max, age=38, salary=10000.0}

Вопросы

IT-Academy

Sorting the Results

IT-Academy

```
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
Root<Employee> emp = criteria.from(Employee.class);
criteria.select(emp).orderBy(
        cb.desc(emp.get("salary")),
        cb.asc(emp.get("name"))
);

List<Employee> employees = em.createQuery(criteria).getResultList();
employees.forEach(System.out::println);
```

select employee0_.id as id1_2_, employee0_.age as age2_2_, employee0_.DEPARTMENT_ID as DEPARTME5_2_, employee0_.name as name3_2_, employee0_.salary as salary4_2_
from Employee employee0_
order by employee0_.salary desc, employee0_.name asc

Employee{id=2, name='Yuli, age=27, salary=16000.99}
Employee{id=5, name='Gleb, age=37, salary=15000.0}
Employee{id=4, name='Paul, age=43, salary=15000.0}
Employee{id=6, name='Li, age=62, salary=13099.0}
Employee{id=3, name='Max, age=38, salary=10000.0}
Employee{id=7, name='Alex, age=22, salary=4500.0}
Employee{id=8, name='null, age=18, salary=300.0}

Projections & Aggregations

IT-Academy

Проекция на количество записей в таблице Employee

```
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Long> criteria = cb.createQuery(Long.class);
criteria.select(cb.count(criteria.from(Employee.class)));
long count = em.createQuery(criteria).getSingleResult();
System.out.println(count);
```

select count(employee0_.id) as col_0_0_ from Employee employee0_

7

Projections & Aggregations

IT-Academy

Проекция на среднее значение поля salary avg()

```
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Double> criteria = cb.createQuery(Double.class);
criteria.select(cb.avg(criteria.from(Employee.class).get("salary")));
double count =  em.createQuery(criteria).getSingleResult();
System.out.println(count);
```

select avg(employee0_.salary) as col_0_0_ from Employee employee0_

10557.141428571427

Projections & Aggregations

IT-Academy

Проекция на количество уникальных значений поля «name» countDistinct()

```
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Long> criteria = cb.createQuery(Long.class);
criteria.select(cb.countDistinct(criteria.from(Employee.class)));
long count = em.createQuery(criteria).getSingleResult();
System.out.println(count);
```

select count(distinct employee0_.id) as col_0_0_ from Employee employee0_

7

Projections & Aggregations

IT-Academy

Проекция на максимальное значений поля «age» - max()

```
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Double> criteria = cb.createQuery(Double.class);
criteria.select(cb.max(criteria.from(Employee.class).get("salary")));
double count =  em.createQuery(criteria).getSingleResult();
System.out.println(count);
```

select max(employee0_.salary) as col_0_0_ from Employee employee0_

16000.99

Projections & Aggregations

IT-Academy

Проекция на минимальное значений поля «salary» - min()

```
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Double> criteria = cb.createQuery(Double.class);
criteria.select(cb.min(criteria.from(Employee.class).get("age")));
Number count =  em.createQuery(criteria).getSingleResult();
System.out.println(count);
```

select min(employee0_.age) as col_0_0_ from Employee employee0_

18

Projections & Aggregations

IT-Academy

Проекция на сумму значений поля «salary» sum()

```
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Double> criteria = cb.createQuery(Double.class);
criteria.select(cb.sum(criteria.from(Employee.class).get("salary")));
Number count =  em.createQuery(criteria).getSingleResult();
System.out.println(count);
```

select sum(employee0_.salary) as col_0_0_ from Employee employee0_

73899.98999999999

Вопросы

IT-Academy

Join

IT-Academy

```
@Test
public void joinTest() {
    EntityManager em = HibernateUtil.getEntityManager();
    CriteriaBuilder cb = em.getCriteriaBuilder();
    CriteriaQuery<Department> criteria = cb.createQuery(Department.class);
    Root<Department> department = criteria.from(Department.class);
    Join<Department, Employee> employeeJoin =
        department.join("employees", JoinType.INNER);
    criteria.where(cb.equal(employeeJoin.get("name"), "Yuli"));
    List<Department> departments = em.createQuery(criteria).getResultList();
    departments.forEach(System.out::println);
}
```

```
select department0_.id as id1_0_, department0_.name as name2_0_
from Department department0_
inner join Employee employees1_ on department0_.id=employees1_.DEPARTMENT_ID
where employees1_.name=?
```

```
Department(id=1, name=Developer,
        employees=[Employee{id=2, name='Yuli, age=27, salary=16000.99},
                Employee{id=3, name='Max, age=38, salary=10000.0},
                Employee{id=4, name='Paul, age=43, salary=15000.0}])
```

Fetch

IT-Academy

```
Integer age = 40;
EntityManager em = HibernateUtil.getEntityManager();
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> criteria = cb.createQuery(Employee.class);
Root<Employee> employee = criteria.from(Employee.class);
employee.fetch("department");
ParameterExpression<Integer> ageParameter = cb.parameter( Integer.class );
criteria.where(cb.gt(employee.get("age"), ageParameter));
List<Employee> employees = em.createQuery(criteria)
        .setParameter(ageParameter, age)
        .getResultList();
employees.forEach(System.out::println);
```

select employee0_.id as id1_1_0_, department1_.id as id1_0_1_, employee0_.age as age2_1_0_, employee0_.DEPARTMENT_ID as DEPARTME5_1_0_, employee0_.name as name3_1_0_, employee0_.salary as salary4_1_0_, department1_.name as name2_0_1_
from Employee employee0_
inner join Department department1_ on employee0_.DEPARTMENT_ID=department1_.id
where employee0_.age>40

Employee{id=4, name='Paul, age=43, salary=15000.0
    Department{id=1, name='Developer'}}
Employee{id=7, name='Li, age=62, salary=13099.0
    Department{id=5, name='QA'}}

Вопросы

IT-Academy

IT-Academy

Спасибо за внимание