JPA Criteria builder IN clause query

(source: java - JPA Criteria builder IN clause query - Stack Overflow)

1. How to write criteria builder api query for below given JPQL query? I am using JPA 2.2.

```
SELECT *
FROM Employee e
WHERE e.Parent IN ('John', 'Raj')
ORDER BY e.Parent
```

2. This criteria set-up should do the trick:

```
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaQuery<Employee> q = cb.createQuery(Employee.class);



Root<Employee> root = q.from(Employee.class);
q.select(root);

List<String> parentList = Arrays.asList(new String[]{"John", "Raj"});

Expression<String> parentExpression = root.get(Employee_.Parent);
Predicate parentPredicate = parentExpression.in(parentList);
q.where(parentPredicate);
q.orderBy(cb.asc(root.get(Employee_.Parent));

q.getResultList();
```

I have used the overloaded CriteriaQuery.where method here which accepts a Predicate.. an in predicate in this case.

3. You can also do this with Criteria API In cluase as below:

```
CriteriaBuilder cb = entityManager.getCriteriaBuilder();
CriteriaQuery<Employee> cq = cb.createQuery(Employee.class);
Root<Employee> root = cq.from(Employee.class);
List<String> parentList = Arrays.asList("John", "Raj");
In<String> in = cb.in(root.get(Employee_parent));
parentList.forEach(p -> in.value(p));

return entityManager
     .createQuery(cq.select(root)
     .where(in).orderBy(cb.asc(root.get(Employee_.Parent)))
     .getResultList();
```

Checkout my Github for this and almost all possible criteria examples.

```
package com.katariasoft.technologies.jpaHibernate.entity.criteria;

import java.math.BigDecimal;
import java.time.Instant;
import java.util.Arrays;

import javax.persistence.criteria.CriteriaBuilder;
import javax.persistence.criteria.CriteriaBuilder.In;
import javax.persistence.criteria.CriteriaQuery;
import javax.persistence.criteria.Root;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

import com.katariasoft.technologies.jpaHibernate.college.data.entity.Instructor;
import com.katariasoft.technologies.jpaHibernate.college.data.entity.Instructor_;
import com.katariasoft.technologies.jpaHibernate.college.data.utils.CriteriaUtils;
import com.katariasoft.technologies.jpaHibernate.college.data.utils.DataPrinters;
import com.katariasoft.technologies.jpaHibernate.college.data.utils.QueryExecutor;

@RunWith(SpringRunner.class)
@SpringBootTest
public class SingleTableCriteriaFetchTests {
	@Autowired
	private CriteriaUtils criteriaUtils;
	@Autowired
	private QueryExecutor queryExecutor;

	@Test
	public void fetchAllInstructors() {
		CriteriaQuery<Instructor> cq = criteriaUtils.criteriaQuery(Instructor.class);
		printResultList(cq.select(cq.from(Instructor.class)));
	}

	@Test
	public void findAllHavingSalaryGreaterThan() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
		Root<Instructor> root = cq.from(Instructor.class);
		printResultList(cq.select(root).where(cb.greaterThan(root.get(Instructor_.salary), BigDecimal.valueOf(30000))));
	}

	@Test
	public void findAllHavingAddressLike() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
		Root<Instructor> root = cq.from(Instructor.class);
		printResultList(cq.select(root).where(cb.like(root.get(Instructor_.address), "%#1074%")));
	}

	@Test
	public void findAllHavingFatherNameLike() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
		Root<Instructor> root = cq.from(Instructor.class);
		printResultList(cq.select(root).where(cb.like(root.get(Instructor_.fatherName), "%Nare%")));

	}

	@Test
	public void findAllHavingFatherNameNotLike() {
```

```
                CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
                CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
                Root<Instructor> root = cq.from(Instructor.class);
                printResultList(cq.select(root).where(cb.notLike(root.get(Instructor_.fatherName), "%Nare%")));
        }

        @Test
        public void findAllHavingAddressAndFatherNameLike() {
                CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
                CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
                Root<Instructor> root = cq.from(Instructor.class);
                printResultList(cq.select(root)
                                .where(cb.greaterThan(root.get(Instructor_.salary), BigDecimal.ZERO),
                                                cb.like(root.get(Instructor_.fatherName), "%nare%"))
                                .orderBy(cb.asc(root.get(Instructor_.birthDateTime)), cb.desc(root.get(Instructor_.name))));

        }

        @Test
        public void findAllHavingAddressAndFatherNameLikeOrSalaryLessThan() {
                CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
                CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
                Root<Instructor> root = cq.from(Instructor.class);
                printResultList(cq.select(root)
                                .where(cb.or(
                                                cb.and(cb.greaterThan(root.get(Instructor_.salary), BigDecimal.ZERO),
                                                                cb.like(root.get(Instructor_.fatherName), "%nare%")),
                                                cb.lessThan(root.get(Instructor_.salary), BigDecimal.valueOf(20000))))
                                .orderBy(cb.asc(root.get(Instructor_.birthDateTime)), cb.asc(root.get(Instructor_.name))));

        }

        @Test
        public void findAllOrderByBirthDateTimeDesc() {
                CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
                CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
                Root<Instructor> root = cq.from(Instructor.class);
                printResultList(cq.select(root).orderBy(cb.desc(root.get(Instructor_.birthDateTime))));
        }

        @Test
        public void findAllBornBetweenBirthDateTimesOrderByBirthDateTimeDesc() {
                CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
                CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
                Root<Instructor> root = cq.from(Instructor.class);
                printResultList(cq.select(root)
                                .where(cb.between(root.get(Instructor_.birthDateTime), Instant.parse("1970-08-
15T16:30:30.000001Z"),
                                                Instant.parse("1993-09-19T14:27:29.999999Z")))
                                .orderBy(cb.desc(root.get(Instructor_.birthDateTime))));

        }

        @Test
        public void findAllHavingIdsIn() {
                CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
                CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
                Root<Instructor> root = cq.from(Instructor.class);
                In<Integer> in = cb.in(root.get(Instructor_.id));
                Arrays.asList(1, 2, 3).forEach(i -> in.value(i));
                printResultList(cq.select(root).where(in));
        }

        @Test
```

```
	public void findAllHavingIdsNotIn() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
		Root<Instructor> root = cq.from(Instructor.class);
		In<Integer> in = cb.in(root.get(Instructor_.id));
		Arrays.asList(1, 2, 3).forEach(i -> in.value(i));
		printResultList(cq.select(root).where(cb.not(in)));
	}

	@Test
	public void findAllNotBornBetweenBirthDateTimesOrderByBirthDateTimeDesc() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<Instructor> cq = cb.createQuery(Instructor.class);
		Root<Instructor> root = cq.from(Instructor.class);
		printResultList(cq.select(root)
				.where(cb.not(cb.between(root.get(Instructor_.birthDateTime),
						Instant.parse("1970-08-15T16:30:30.000001Z"), Instant.parse("1993-09-
19T14:27:29.999999Z"))))
				.orderBy(cb.desc(root.get(Instructor_.birthDateTime))));
	}

	@Test
	public void findNameAndSalaryHavingSalaryGreterThan() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<Object[]> cq = cb.createQuery(Object[].class);
		Root<Instructor> root = cq.from(Instructor.class);
		printResultList(cq.multiselect(root.get(Instructor_.name), root.get(Instructor_.salary))
				.where(cb.greaterThan(root.get(Instructor_.salary), BigDecimal.valueOf(75000))));
	}

	@Test
	public void findDistinctFatherName() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<String> cq = cb.createQuery(String.class);
		Root<Instructor> root = cq.from(Instructor.class);
		printResultList(cq.select(root.get(Instructor_.fatherName)).distinct(true));
	}

	@Test
	public void countHavingSalaryBetween() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<Long> cq = cb.createQuery(Long.class);
		Root<Instructor> root = cq.from(Instructor.class);
		printValue((cq.select(cb.count(root))
				.where(cb.between(root.get(Instructor_.salary), BigDecimal.ZERO, BigDecimal.valueOf(100000.00)))));
	}

	// TODO
	@Test
	public void findMinSalary() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<BigDecimal> cq = cb.createQuery(BigDecimal.class);
		Root<Instructor> root = cq.from(Instructor.class);
		printValue(cq.select(cb.min(root.get(Instructor_.salary))));
	}

	@Test
	public void findMaxSalary() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<BigDecimal> cq = cb.createQuery(BigDecimal.class);
		Root<Instructor> root = cq.from(Instructor.class);
		printValue(cq.select(cb.max(root.get(Instructor_.salary))));
	}
```

```
	@Test
	public void calculateAverageSalary() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<Double> cq = cb.createQuery(Double.class);
		Root<Instructor> root = cq.from(Instructor.class);
		printValue(cq.select(cb.avg(root.get(Instructor_.salary))));
	}

	@Test
	public void countHavingFatherName() {
		CriteriaBuilder cb = criteriaUtils.criteriaBuilder();
		CriteriaQuery<Object[]> cq = cb.createQuery(Object[].class);
		Root<Instructor> root = cq.from(Instructor.class);
		printResultList(cq.multiselect(root.get(Instructor_.fatherName),
cb.count(root)).groupBy(root.get(Instructor_.fatherName))
				.having(cb.gt(cb.count(root), 0)));
	}

	private <T> void printResultList(CriteriaQuery<T> criteriaQuery) {
		DataPrinters.listDataPrinter.accept(queryExecutor.fetchListForCriteriaQuery(criteriaQuery));
	}

	private <T> void printValue(CriteriaQuery<T> criteriaQuery) {
		System.out.println(queryExecutor.fetchValueForCriteriaQuery(criteriaQuery));
	}

}
```

4.

```
List<String> parentList = Arrays.asList("John", "Raj");
final CriteriaBuilder cb = entityManager.getCriteriaBuilder();
final CriteriaQuery<Employee> query = cb.createQuery(Employee.class);
final Root<Employee> employee = query.from(Employee.class);

query.select(employee).where(employee.get("Parent").in(parentList));
```

This should work fine. For more info please refer this article by baeldung. It is very resourceful.

(1) Overview

We often come across problems where we need to query entities based on whether a single-valued attribute is a member of a given collection.

In this tutorial, we'll learn how to solve this problem with the help of the Criteria API.

(2) Sample Entities

Before we start, let's take a look at the entities that we're going to use in our write-up.

We have a DeptEmployee class that has a many-to-one relationship with a Department class:

```
@Entity
public class DeptEmployee {
   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   private long id;

   private String title;

   @ManyToOne
   private Department department;
}
```

Also, the Department entity that maps to multiple DeptEmployees.

```
@Entity
public class Department {
   @Id
   @GeneratedValue(strategy = GenerationType.SEQUENCE)
   private long id;

   private String name;

   @OneToMany(mappedBy="department")
   private List<DeptEmployee> employees;
}
```

(3) The CriteriaBuilder.in

First of all, let's use the CriteriaBuilder interface.

It can be used to test whether the given expression is contained in the list of values:

```
CriteriaQuery<DeptEmployee> criteriaQuery =
 criteriaBuilder.createQuery(DeptEmployee.class);
Root<DeptEmployee> root = criteriaQuery.from(DeptEmployee.class);
In<String> inClause = criteriaBuilder.in(root.get("title"));
for (String title : titles) {
   inClause.value(title);
}
criteriaQuery.select(root).where(inClause);
```

(4) The Expression.in

Alternatively, we can use a set of overloaded in() methods from the Expression interface:

```
criteriaQuery.select(root)
 .where(root.get("title")
 .in(titles));
```

As we can see it also simplifies our code a little bit.

(5) IN Expressions Using Subqueries

So far, we have used collections with predefined values. Now, let's take a look at an example when a collection is derived from an output of a subquery.

For instance, we can fetch all DeptEmployees who belong to a Department, with the specified keyword in their name:

```
Subquery<Department> subquery = criteriaQuery.subquery(Department.class);
Root<Department> dept = subquery.from(Department.class);
subquery.select(dept)
  .distinct(true)
  .where(criteriaBuilder.like(dept.get("name"), "%" + searchKey + "%"));

criteriaQuery.select(emp)
  .where(criteriaBuilder.in(emp.get("department")).value(subquery));
```

Here, we created a subquery that was then passed into the value() as an expression to search for the Department entity.

(6) Conclusion

In this quick article, we have learned different ways to achieve the IN operation using the Criteria API. We have also explored how to use the Criteria API with subqueries.

Finally, the complete implementation for this tutorial is available on GitHub.

5. I hope it could be useful. The code tested in case where Entry has only one @Id column:

```
public static <Entry, Id> List<Entry> getByIds(List<Id> ids, Class<Entry> clazz, EntityManager entityManager) throws
CustomDatabaseException
{
   List<Entry> entries;
   try
   {
      CriteriaBuilder cb = entityManager.getCriteriaBuilder();
      CriteriaQuery<Entry> q = cb.createQuery(clazz);
      Root<Entry> root = q.from(clazz);
      EntityType<Entry> e = root.getModel();
      CriteriaQuery<Entry> all = q.where(root.get(e.getId(e.getIdType().getJavaType()).getName()).in(ids));

      TypedQuery<Entry> allQuery = entityManager.createQuery(all);
      entries = allQuery.getResultList();
   }
   catch (NoResultException nre)
   {
      entries = Collections.emptyList()
   }
   catch (Exception e)
   {
      throw new CustomDatabaseException(e);
   }
   return entries;
}
```