1. **@ManyToOne**

@Entity

public class **Address** {

@Id @GeneratedValue

private Long id;

private String street;

private String city;

private String zipCode;

@ManyToOne

private User owner;

}

public enum Gender {

MALE,

FEMALE,

NOT\_SURE

}

@Embeddable

public class Login {

private String username;

private String password;

}

@Entity

public class **User** {

@Id @GeneratedValue

private Long id;

private String firstName;

private String lastName;

private int credits;

private Login login;

@OneToMany(cascade = CascadeType.PERSIST, mappedBy = "owner")

private Set<Address> addresses = new HashSet<Address>();

@Temporal(TemporalType.DATE)

private Date birthDate;

@Enumerated(EnumType.STRING)

private Gender gender;

}

@Transactional

public class UserManager {

//get all users living in Paris

@Transactional

public List<User> getUsersFromParis() {

EntityManager em = lazyEM.get();

CriteriaBuilder cb = em.getCriteriaBuilder();

final CriteriaQuery<**User**> cr = cb.createQuery( User.class );

final Root<**User**> root = cr.from( User.class );

final SetJoin<User,Address> a = root.join(User\_.addresses ); // мы получили объект “a” из которого

// уже будем выбирать поля сущности “Address”

root.fetch(User\_.addresses ); //это доп.настройка, говорящая, что поля разных сущностей

// должны “выгребаться” одним запросом

cr.select(root).distinct( true ).where( cb.equal( a.get( Address\_.city ), "Paris" ) );

final TypedQuery<User> typedQuery = em.createQuery( cr );

typedQuery.setFirstResult( 0 ).setMaxResults( 20 );

final List<User> resultList = typedQuery.getResultList();

return resultList;

}

**Пример с каскадом связанных таблиц**

**(получаем столбец из нижней таблицы)**

**public** **static** Specific<JpaAction> hasArtifact(**final** String controllerId, **final** String sha1Hash) {

**return** (actionRoot, query, criteriaBuilder) -> {

Join<Action, DistrSet> **dsJoin** = **root**.join(Action\_.distrnSet);

SetJoin<DistrSet, SoftwareModule> **modulesJoin** = **dsJoin**.join(DistrSet\_.modules);

ListJoin<SoftwareModule, Artifact> **artifactsJoin** = **modulesJoin**.[join](https://www.codota.com/code/java/methods/javax.persistence.criteria.SetJoin/join)(SoftwareModule\_.artifacts);

**return** criteriaBuilder.and

(criteriaBuilder.equal(**artifactsJoin**.get(JpaArtifact\_.sha1Hash), sha1Hash),

criteriaBuilder.equal(**root**.get(JpaAction\_.target).get(JpaTarget\_.controllerId),

controllerId));

};

}

**2. @ManyToMany**

У меня есть 2 класса Answer и Collaborator, в отношении @ManyToMany.

class Answer {

@ManyToMany(cascade = CascadeType.ALL)

@JoinTable(name = "ANSWERS\_COLLABORATORS", joinColumns = { @JoinColumn(name = "aid") }, inverseJoinColumns = { @JoinColumn(name = "cid") })

private Set<Collaborator> collaborators = new HashSet<Collaborator>(0);

}

Класс Answer имеет набор Collaborator, но Collaborator не хранит набор Answer. Что мне нужно сделать в Hibernate, так CriteriaQuery это найти соавторов для ответа, заданного id.

Я уже сделал это с помощью Hibernate Criteria( org.hibernate.Criteria) с помощью преобразователя результатов, но я застрял, когда дело доходит до использования CriteriaQuery, потому что у меня нет списка ответов для соединения.

Наконец-то готово ...

Вот код:

public List<Collaborator> getCollaborators(Long answerId) {

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<**Collaborator**> cr = cb.createQuery(Collaborator.class);

Root<**Answer**> root = cr.from(Answer.class);

cr.where(cb.equal(root.get(Answer\_.id), answerId));

SetJoin<Answer, Collaborator> answers = root .join(Answer\_.collaborators);

CriteriaQuery<Collaborator> cq = cr.select(answers);

TypedQuery<Collaborator> query = em.createQuery(cr);

return query.getResultList();

}

---------------------------------------------------------------------------------------------

Используя JPA2 Criteria:

CriteriaBuilder cb = em.getCriteriaBuilder();

CriteriaQuery<**Long**> cr = cb.createQuery(Collaborator.class);

Root<**Answer**> root = cr.from(Answer.class);

Join<Collaborator,Answer> joinAnswerCollaborator = root.join("Answer\_collaborators");

Используя HQL Criteria:

Вы можете использовать это:

Criteria criteria = session.createCriteria(Answer.class);

criteria.setResultTransformer(Criteria.DISTINCT\_ROOT\_ENTITY);

criteria.createAlias("collaborators", "collaborators");

criteria.add(Restrictions.eq("collaborators.id",desiredCollaboratorId);

чтобы получить все ответы, связанные с определенным соавтором.

И это:

Criteria criteria = session.createCriteria(Answer.class);

criteria.setResultTransformer(Criteria.DISTINCT\_ROOT\_ENTITY);

criteria.setFetchMode("collaborators", FetchMode.JOIN)

criteria.add(Restrictions.idEq(desiredAnswerId));

dsrTrackingCriteria.setProjection(Projections.property("collaborators"));

чтобы связать всех соавторов с определенным ответом.

1. **@ManyToMany**

Добрый день! Есть две сущности с отношением Many to Many:

1. @Entity
2. @Table(name = "items")
3. public class Item implements Comparable {
5. @Id
6. @GeneratedValue(strategy = GenerationType.IDENTITY)
7. @Column(name = "item\_id")
8. private Integer itemId;
9. ....
10. @ManyToMany(cascade = CascadeType.PERSIST, fetch = FetchType.LAZY)
11. @JoinTable(name = "items\_criteria",
12. joinColumns = @JoinColumn(name = "item\_id"),
13. inverseJoinColumns = @JoinColumn(name = "filter\_criterion\_id"))
14. private List<FilterCriterion> filterCriteria;
15. ....
16. }

и

1. @Entity
2. @Table(name = "filter\_criteria")
3. public class FilterCriterion {
5. @Id
6. @GeneratedValue(strategy = GenerationType.IDENTITY)
7. @Column(name = "filter\_criterion\_id")
8. private Integer filterCriterionId;
9. ....
10. @ManyToMany(cascade = CascadeType.PERSIST, fetch = FetchType.LAZY)
11. @JoinTable(name = "items\_criteria",
12. joinColumns = @JoinColumn(name = "filter\_criterion\_id"),
13. inverseJoinColumns = @JoinColumn(name = "item\_id"))
14. private List<Item> items;
15. ....

В классе ItemDao нужно реализовать метод, который будет принимать в качестве аргумента коллекцию FilterCriterion и возвращать коллекцию Item, такую, чтобы у каждого Item в этой коллекции, коллекция FilterCriterion содержала все элементы коллекции, данной в качестве аргумента. При такой реализации:

1. @Override
2. public List<Item> getItems(List<FilterCriterion> currentFilterCriteria) {
4. Criteria criteria = ht.getSessionFactory().getCurrentSession().createCriteria(Item.class);
5. List<Integer>currentFilterCriteriaId = new ArrayList<Integer>();
6. for(FilterCriterion criterion : currentFilterCriteria){
7. currentFilterCriteriaId.add(criterion.getFilterCriterionId());
8. }
9. if(!currentFilterCriteriaId.isEmpty()){
10. criteria.createAlias("filterCriteria", "f");
11. criteria.add(Restrictions.in("f.filterCriterionId", currentFilterCriteriaId));
12. }
13. return criteria.list();
14. }

в результат попадают все Item, у которых хотя бы один FilterCriterion имеется в коллекции, данной в качестве аргумента. Если

1. criteria.add(Restrictions.in("f.filterCriterionId", currentFilterCriteriaId));

заменить на

1. for(Integer currentId : currentFilterCriteriaId){
2. criteria.add(Restrictions.eq("f.filterCriterionId", currentId));
3. }

результат все равно неверный. Как правильно сформировать criteria, чтобы в результате получить только те Item, коллекции FilterCriterion которых содержат все элементы данной в качестве аргумента коллекции FilterCriterion?

Целый день ломал голову над данным вопросом, правда использовал JPQL (но думаю, что его можно переделать в Criteria API).  
Делал на своём примере: есть сущности Book и Author. Между ними связь многие ко многим (посредством связующей таблицы Authors\_Books).  
Код:

1. @Override
2. public List<Book> getBooksByAuthors(List<Author> authors)
3. {
4. TypedQuery<Book> query = entityManager.createQuery("select b from Book b join b.authors ba where size(b.authors) = :authors\_count and ba in (:authors)group by b having count (b) = :authors\_count", Book.class);
6. query.setParameter("authors\_count", authors.size());
7. query.setParameter("authors", authors);
9. return query.getResultList();
10. }

Тоже самое, но с использованием JPA Criteria API:

1. @Override
2. public List<Book> getBooksByAuthors(List<Author> authors)
3. {
4. CriteriaBuilder cb = em.getCriteriaBuilder();
5. CriteriaQuery<Book> cr = cb.createQuery(Book.class);
6. Root<Book> root = cr.from(Book.class);
7. ListJoin<Book, Author> joinBookAuthors = root.join(Book\_.authors);
9. Expression<List<Author>> bookAuthors = root.get(Book\_.authors);
10. Expression<Integer> countOfBookAuthors = cb.size(bookAuthors);
11. Expression<Long> countOfBooksInGroup = cb.count(root);
13. Predicate predicateCountOfBookAuthorsEqualsInputListSize = cb.equal(countOfBookAuthors, authors.size());
14. Predicate predicateBookAuthorsInInputList = joinBookAuthors.in(authors);

17. cr.select(root).where(cb.and(predicateCountOfBookAuthorsEqualsInputListSize, predicateBookAuthorsInInputList))
18. .groupBy(rootBook)
19. .having(cb.equal(countOfBooksInGroup, authors.size()));
21. return em.createQuery(query).getResultList();
22. }

Код сущностей:

1. @Entity
2. public class Book
3. {
4. @Id
5. @GeneratedValue(strategy = GenerationType.IDENTITY)
6. private Long id;
7. private Float price;
8. private String isbn;
9. private Boolean illustration;
11. @NotNull
12. @Column(nullable = false, updatable = false)
13. private String title;
14. @Size(min = 5, max = 200)
15. private String description;
17. @Column(nullable = false)
18. private Integer nbOfPage;
19. @ElementCollection(fetch = FetchType.LAZY)
20. @CollectionTable(name = "tag")
21. @Column(name = "value")
22. private List<String> tags;
23. @ManyToMany(cascade = {CascadeType.PERSIST})
24. @JoinTable(name = "authors\_books",
25. joinColumns = @JoinColumn(name = "book\_id"),
26. inverseJoinColumns = @JoinColumn(name = "author\_id"))
27. private List<Author> authors = new ArrayList<>();
28. }
29. @Entity
30. public class Author
31. {
32. @Id
33. @GeneratedValue(strategy = GenerationType.IDENTITY)
34. private Long id;
35. @NotNull
36. @Size(min = 3, max = 30)
37. @Column(length = 30, nullable = false)
38. private String name;
39. @ManyToMany(mappedBy = "authors")
40. @OrderBy("title ASC")
41. private List<Book> books = new ArrayList<>();
42. }