Hibernate Criteria Extension

HCE 1.0.1
Beta Version

EASY AND FAST DEVELOPMENT

USING HCE, SIMPLE TO DO

BASED ON HIBERNATE CRITERIA AND PROJECTIONS

HCE 1.0.1 References

BY : KISMAN HONG
Chief Information Analyst (CIA)

Table of contents

About the Author

Kisman Hong is a Java developer. Working at Bank for the first experience as a developer of Internet Banking project. Using struts java framework in many java web based project. Now working at Infoflow Solutions as "Software Specialist" developing and enhance "Collection Management System".

Author of "Kismiss Report" that generate jasper xml template through annotation in POJO.

Now, create this Hibernate Criteria Extension for simplifying code and reducing development time.

Preface

Nowadays, IT projects held in almost every business part of the world. Almost all business adopt technology system for helping them s o lv i n g the complexity of their jobs. Many IT service companies grown up and offer many solutions to simplify business complexity. IT service company need to choose a best programming that should be adopted. Today, IT projects are written in many kinds of "programming language" such as Java, Microsoft .Net, Visual Basic, Delphi, Phyton, etc.

Java is the most reliable programming language used for developing core project. Million of IT projects have been built using Java. Therefore, Java is a widely used programming language. Many Java framework introduced for helping developer / programmer to develope Java applications. Many frameworks come out to meet the needs.

One of the most important issue when developing application is development time. The longer we develop a project, the project cost will be up. Development time is not the only issue that must be concerned, we must sure the quality of code, code effectiveness, and future maintenance. For these issues, many developers use hibernate to simplify their code. Hibernate is an Object Relation Mapping (ORM) tool to help us query data from data source and return the data directly to object. Hibernate is a great tool, widely used, and easy documentation. Hibernate save our development time. We can reduce many codes.

Hibernate has performance issue. When talking hibernate, it's all about object. So everything will be returned as object or collection of objects. It means that if we let hibernate querying data from data source, we will have everything of the entity and it's assoction(s) based on configuration of the entity. If we want to get only part of the object, we need to use projections for defining the columns to be selected. I think an object will be used for many conditions, sometimes we just need one column or some columns, we cannot let hibernate get all of them for everytime, this will cause performance cost. For a small object this will not be an issue, but I think it is better for us to get only the columns that we need. By using hibernate projections API, we can do projections. If we have many entities that need to use projections, we need to write method as many as number of entities. Object that has association(s) will give a complicated issue. Remember that projections of association in hibernate cannot be returned to the object. We need to declare new property to get every column value from the associated object.

Hibernate Criteria Extension (HCE) is written to solve hibernate limitation. HCE can simplify your code. HCE API can let you to select part of columns that you need, we just give the field names that we want to select. HCE support for association and automatically detect join type from entity mapping. HCE is written based on hibernate criteria and projections.

HCE offer you a better way to generate query script without spend too much time. HCE give you a simple way to declare select, restriction, and order.

HCE's goal is to relieve developer a better way to select columns without ignore the performance issue. HCE may not be the best solution for every conditions. However, HCE can certainly help you to simplify projections declaration, restrictions with association, and order.

Why Hibernate Criteria Extension (HCE)?

HCE give us some reasons on why it should be used:

- I suggest that we should get the data only that we need. Although for this time there is not performance issue when selecting a table, we have never know for future if it still make sense for us to get all the column data at one time that we don't need all of them absolutely (not every field is used). In Hibernate, we can use projections. Using projections

need more code to do.

- Let hibernate select all columns in a entity causing debug problem. We will be confused if we only want to track one column value. We need to substract the query generated before copy and paste to database browser.
- Using criteria and projections prevent n+1 select problem. By using HCE and give the projections, we will override the default configuration of hibernate, the columns to be selected is chosen well.
- The less code you write, the less time you need to build an application. By using HCE, I hope we reduce to write method. We can use one method for many solutions.

If you are interested for using HCE, There are some steps that must be followed:

- If you are new to hibernate, you must understand it before you begin to use HCE.
- HCE is written based on hibernate criteria and projections.
- The hibernate column mapping must be used on the field, cannot support for getter method mapping.
- Use this references documentation as your primary source information. We will give the example in application.

Hopefully, HCE users can be fulfilled to simplify hibernate criteria and projections. Thank you for using HCE.

CHAPTER 1

Introduction to Hibernate Criteria Extension

HCE

Hibernate Criteria Extension (HCE) is used for helping us to simplify our criteria query. HCE will automatically detect join and only get the data column(s) that is needed. We can define column(s) to be selected only by passing the field name. Restrictions and orders can be easyly defined only by passing Expression class in HCE API. By using HCE, we can simplify our code and write less code than the usual. "One method for more" concept that mean we can write one method to be used for many situations.

1.1 Preface

This chapter is an introductory tutorial for new users of Hibernate Criteria Extension (HCE). Before using HCE, we need to understand hibernate well.

This tutorial is intended for new users of HCE that require hibernate knowledge.

HCE is based on hibernate criteria and projections.

1.2. HCE Architecture

HCE is a third party library that is written based on hibernate criteria and projections. We can easyly query data from data source.

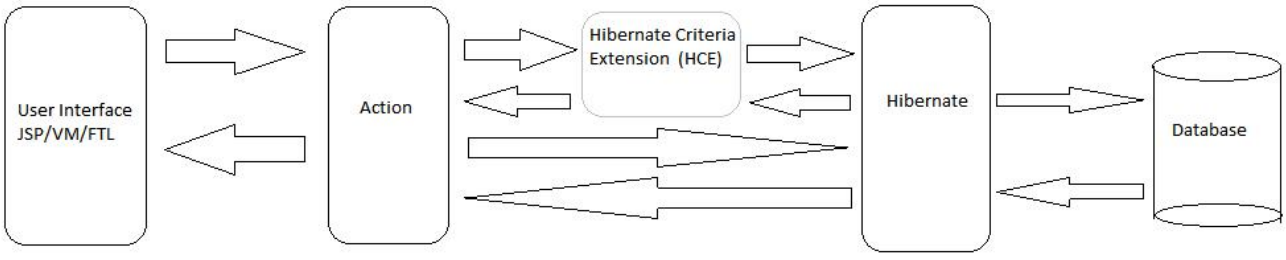
The first thing we need is including the HCE required libraries in application, shown like below :

.
+ lib
log4j.jar
slf4j-api.jar slf4j-
log4j.jar commons-
lang.jar commons-
logging.jar
hibernate-core.jar
javax.persistence.jar
servlet-api.jar
dom4j.jar

HCE's features :

- Declare hibernate projections by passing field(s) name, even an association. (select only the needed field(s) / column(s)).
- Declare restriction(s) by passing "Expression"; Expression contains : propertyName, restrictionType, value. This is valid for association.
- Declare order(s) by passing Order; Order contains : propertyName, orderType. This is valid for association.

Here is HCE's Architecture :



Hibernate Criteria Extension Architecture

HCE help us to create association(s), filter(s), or order(s) before processed by hibernate. After

getting data from database, hibernate ResultTransformer will be replaced by HCEResultTransformer that implements ResultTransformer. Remember that HCE help us to create association(s), filter(s), or order(s) by informations given, so we do not need to create join (alias or criteria even sub criteria) manually when using projections.

CHAPTER 2

Uses of Hibernate Criteria Extension

HCE

This section provides the essential uses of HCE. We will discuss how to simplify code by using HCE.

HCE API will be discussed in this section.

2.1. Preface

This chapter is an introductory uses of HCE. HCE examples will be shown. Once again, hibernate knowledge is required before using HCE.

2.2. HCE Example

HCE require initialization setting. We need to pass hibernate session factory to HCE. Here is an example HCE configuration :

```
import javax.servlet.ServletContext; import
javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;

import org.apache.commons.logging.Log; import
org.apache.commons.logging.LogFactory;

import softtech.hong.hce.initial.HCESetup;

import com.softtech.hrms.connection.HibernateSessionFactory;

/**
 * The Class ContextListener.
*
 * @author Kisman Hong
 */
public class HCEInitial implements ServletContextListener {

	/** The Constant logger. */
	private static Log logger = LogFactory.getLog(HCEInitial.class);

	/** The context. */
	private ServletContext context = null;

	/**
	 * Record the fact that this web application has been destroyed.
*
	 * @param event
	 *            The servlet context event
	 */
	public void contextDestroyed(ServletContextEvent event)
		{ try {

		} catch (Exception e) {
			log("Nothing happened", e);
		}
		log("Web application shutdown");
		this.context = null;
	}

	/**
	 * Record the fact that this web application has been initialized.
*
	 * @param event
```

```
         * The servlet context event */

        public void contextInitialized(ServletContextEvent event)
                { try {
                        log("==============* HCE Setup *================"); HCESetup hceSetup =

                        new HCESetup() hceSetup.setSessionFactory(

                        HibernateSessionFactory.getSessionFactory());

                        hceSetup.cacheTableNameForEntity();

                } catch (Exception e) {
                        e.printStackTrace();
                        log("Failed to cache entity to table");
                }
        }

        /**
         * Log a message to the servlet context application log.
*
         * @param message
         *            Message to be logged
         */
        private void log(String message) {
                if (context != null) {
                        context.log(message);
                } else {
                        System.out.println("ContextListener: " + message);
                }
        }

        /**
         * Log a message and associated exception to the servlet context application log.
*
         * @param message
         *            Message to be logged
         * @param throwable
         *            Exception to be logged
         */
        private void log(String message, Throwable throwable) {
                logger.error(throwable.getMessage(), throwable);

                if (context != null) {
                        context.log("ContextListener: " + message, throwable);
                } else {
                        System.out.println("ContextListener: " + message);
                        throwable.printStackTrace(System.out);
                }

        }

}
```

Call your code above in web.xml

```
<listener> <listener-
    class>your.package.HCEInitial</listener-class>
</listener>
```

You can also use HCEServlet class that provided by hce. Using HCEServlet is simpler than the other way, just pust a listener to your web.xml :

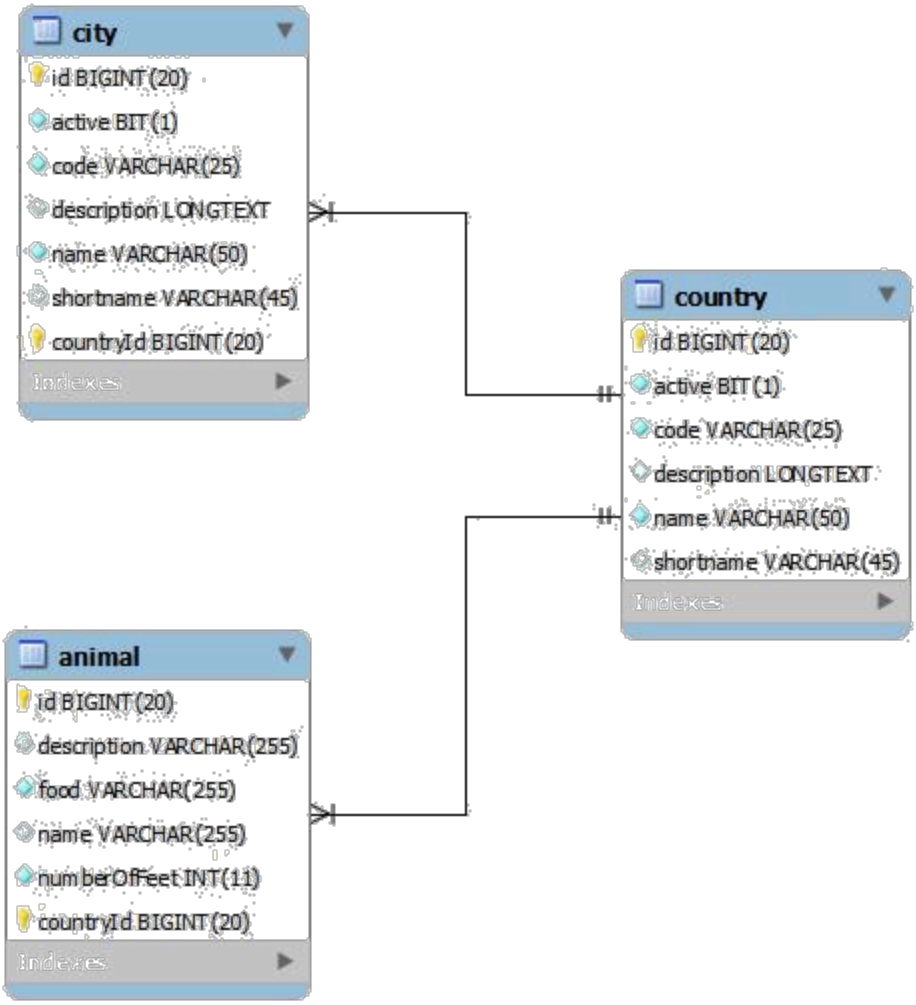
```
<listener> <listener-
    class>softtech.hong.hce.initial.HCEServlet</listener-
    class>
</listener>
```

Spring integration

It is easy for us to initialize HCE in spring context. It can be looked like :

```
<bean id="sessionFactory"
class="org.springframework.orm.hibernate3.annotation.AnnotationSessionFactoryBean">
            <property name="dataSource" ref="dataSource"/>
            <property name="packagesToScan">
                  <list>
                        <value>com.softtech.hrms.model</value>
                  </list>
            </property>
            <property name="hibernateProperties">
                  <props>
                        <prop key="hibernate.dialect">
                              com.softtech.hrms.action.EnhancedMySQL5HibernateDialect
                        </prop>
                        <prop key="hibernate.show_sql">true</prop>
                        <prop key="hibernate.format_sql">true</prop>
                        <prop key="hibernate.connection.autocommit">false</prop>


                        <prop key="hibernate.c3p0.min_size">5</prop>
                        <prop key="hibernate.c3p0.max_size">20</prop>

                        <prop key="hibernate.c3p0.max_statements">180</prop>

                        ...
                  </props>
            </property>
</bean>

<bean id="hce" class="softtech.hong.hce.initial.HCESetup">
      <property name="sessionFactory" ref="sessionFactory" />
</bean>
```

Suppose that we have Animal, Country, and City. The relations between Animal, Country, and Country are shown as below :



And we will create our POJO like :

- **Animal.java**

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

@Entity
public class Animal {

	@Id
	@GeneratedValue
	private Long id;

	private String name;

	@Column(nullable=false)
```

```
private int numberOfFeet;

@Column(nullable=false)
private String food;

private String description;

@ManyToOne(fetch = FetchType.EAGER)
@JoinColumn(name = "countryId", nullable = false, updatable = false)
private Country country;

public Long getId() {
	return id;
}

public void setId(Long id) {
	this.id = id;
}

public String getName() {
	return name;
}

public void setName(String name) {
	this.name = name;
}

public int getNumberOfFeet() {
	return numberOfFeet;
}

public void setNumberOfFeet(int numberOfFeet) {
	this.numberOfFeet = numberOfFeet;
}

public String getFood() {
	return food;
}

public void setFood(String food) {
	this.food = food;
}

public String getDescription() {
	return description;
}

public void setDescription(String description) {
	this.description = description;
}

public Country getCountry() {
	return country;
}

public void setCountry(Country country) {
```

```
            this.country = country;
        }

    }
```

- **Country.java**

```
import java.util.Set;

import javax.persistence.CascadeType;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;
import   javax.persistence.OneToMany;
import javax.persistence.OrderBy;

import org.apache.commons.lang.builder.ToStringBuilder;

@Entity (name= "country")
public class Country {

        @Id
        @GeneratedValue
        private Long id;

        @Column(length = 25, nullable = false)
        private String code;

        @Column(length = 50, nullable = false)
        private String name;

        @Column(length = 1000)
        private String description;

        @Column(length = 45)
        private String shortname;

        @Column(nullable = false, updatable=false)
        private boolean active;

        @OneToMany(mappedBy = "countryId", cascade= CascadeType.ALL, fetch =
                FetchType.LAZY)
        @Column(name = "id")
        @OrderBy("name")
        private Set<City> cities;

        public Long getId()
                { return id;
        }

        public void setId(Long id) {
                this.id = id;
        }
```

```
	public String getCode()
		{ return code;
	}

	public void setCode(String code) {
		this.code = code;
	}

	public String getName()
		{ return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getDescription() {
		return description;
	}

	public void setDescription(String description) {
		this.description = description;
	}

	public String getShortname() {
		return shortname;
	}

	public void setShortname(String shortname) {
		this.shortname = shortname;
	}

	public boolean isActive()
		{ return active;
	}

	public void setActive(boolean active) {
		this.active = active;
	}

	public Set<City> getCities() {
		return cities;
	}

	public void setCities(Set<City> cities)
		{ this.cities = cities;
	}

}
```

- **City.java**

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.FetchType;
import javax.persistence.GeneratedValue;
```

```java
import javax.persistence.Id;
import javax.persistence.JoinColumn;
import javax.persistence.ManyToOne;

import org.apache.commons.lang.builder.ToStringBuilder;

@Entity (name= "city")
public class City {

	@Id
	@GeneratedValue
	private Long id;

	@ManyToOne(fetch = FetchType.EAGER)
	@JoinColumn(name = "countryId", nullable = false, updatable = false)
	private Country countryId;

	@Column(length = 25, nullable = false)
	private String code;

	@Column(length = 50, nullable = false)
	private String name;

	@Column(length = 1000)
	private String description;

	@Column(length = 45)
	private String shortname;

	@Column(nullable=false, updatable=false)
	private boolean active;

	public Long getId()
		{ return id;
	}

	public void setId(Long id)
		{ this.id = id;
	}

	public Country getCountryId()
		{ return countryId;
	}

	public void setCountryId(Country countryId)
		{ this.countryId = countryId;
	}

	public String getCode()
		{ return code;
	}

	public void setCode(String code)
		{ this.code = code;
	}
```

```
	public String getName()
		{ return name;
	}

	public void setName(String name)
		{ this.name = name;
	}

	public String getDescription()
		{ return description;
	}

	public void setDescription(String description)
		{ this.description = description;
	}

	public String getShortname()
		{ return shortname;
	}

	public void setShortname(String shortname)
		{ this.shortname = shortname;
	}

	public boolean isActive()
		{ return active;
	}

	public void setActive(boolean active)
		{ this.active = active;
	}

}
```

Case 1: Select all field from one table.

TODO: select all field from animal.

- **Hibernate Criteria**

```
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.criterion.DetachedCriteria;

import softtech.hong.hce.core.QueryTransformer;
import softtech.hong.hce.junit.connection.HibernateSessionFactory;
import softtech.hong.hce.junit.helper.PageBean;
import softtech.hong.hce.junit.model.Animal;
import softtech.hong.hce.model.Order;

/**
 * @author Kisman Hong
 * this is an example only, for the real, we need more layer such as DAO
 * this code is provided as is, i just want to show how to use hce
```

```
 */
public class AnimalService {

	/**
	 * @param projections -> field or property to be selected
	 * @return List of Animal
	 */
	@SuppressWarnings("unchecked")
	public List<Animal> loadAll(){
		Session session = HibernateSessionFactory.getSession();
		DetachedCriteria detachedCriteria =
			DetachedCriteria.forClass(Animal.class);
		List<Animal> animals =
			detachedCriteria.getExecutableCriteria(session).list();
		return animals;
	}
}
```

The sql query will be looked like :

```
    select
        this_.id as id0_1_,
        this_.countryId as countryId0_1_,
        this_.description as descript2_0_1_,
        this_.food as food0_1_,
        this_.name as name0_1_,
        this_.numberOfFeet as numberOf5_0_1_,
        country2_.id as id1_0_,
        country2_.active as active1_0_,
        country2_.code as code1_0_,
        country2_.description as descript4_1_0_,
        country2_.name as name1_0_,
        country2_.shortname as shortname1_0_
    from
        Animal this_
    inner join
        country country2_
            on this_.countryId=country2_.id
```

- **HCE Query**

```
import java.util.List;

import org.hibernate.Criteria;
import org.hibernate.Session;
import org.hibernate.criterion.DetachedCriteria;

import softtech.hong.hce.core.QueryTransformer;
import softtech.hong.hce.junit.connection.HibernateSessionFactory;
import softtech.hong.hce.junit.helper.PageBean;
import softtech.hong.hce.junit.model.Animal;
import softtech.hong.hce.model.Order;

/**
 * @author Kisman Hong
 * this is an example only, for the real, we need more layer such as DAO
```

```
 * this code is provided as is, i just want to show how to use
 hce */
public class AnimalService extends QueryTransformer<Animal>{

	/**
	 * @param projections -> field or property to be selected
	 * @return List of Animal
	 */
	@SuppressWarnings("unchecked")
	public List<Animal> loadAll(){
		Session session = HibernateSessionFactory.getSession();
		/* select all, let hibernate do it */
		DetachedCriteria detachedCriteria = criteriaWithProjections(null);
			// return DetachedCriteria
		List<Animal> animals =
			detachedCriteria.getExecutableCriteria(session).list();
		return animals;
	}
}
```

The sql query will be looked like :

```
select
    this_.id as id0_1_,
    this_.countryId as countryId0_1_,
    this_.description as descript2_0_1_,
    this_.food as food0_1_,
    this_.name as name0_1_,
    this_.numberOfFeet as numberOf5_0_1_,
    country2_.id as id1_0_,
    country2_.active as active1_0_,
    country2_.code as code1_0_,
    country2_.description as descript4_1_0_,
    country2_.name as name1_0_,
    country2_.shortname as shortname1_0_
from
    Animal this_
inner join
    country country2_
        on this_.countryId=country2_.id
```

At this case, both hibernate criteria and hce query produce the same sql query. HCE passing null as projections parameter that mean select everything based on hibernate configuration.

Case 2: Select one column from one table.

TODO: select only name from animal.

- **Hibernate Criteria**

  code :

  ```
  DetachedCriteria detachedCriteria = DetachedCriteria.forClass(Animal.class);
  detachedCriteria.setProjection(Projections.property("name"));
  ```

  The sql query will be looked like :

```
select
    this_.name as y0_
from
    Animal this_
```

- **HCE Query**

  Code :

```
DetachedCriteria detachedCriteria = criteriaWithProjections(new String[]
{"name"});
```

  The sql query will be looked like :

```
select
    this_.name as y0_
from
    Animal this_
```

By using hibernate, we need to mention "name" and set it to projections. In HCE, the projections is parameter of `criteriaWithProjections` method.

Case 3: Select many columns from one table.

TODO: select id, name, food, numberOfFeet, and description from animal

- **Hibernate Criteria**

  Code :

```
DetachedCriteria detachedCriteria =
      DetachedCriteria.forClass(Animal.class);
      detachedCriteria.setProjection(Projections.projectionList()
            .add(Projections.property("id"))
            .add(Projections.property("name"))
            .add(Projections.property("food"))
            .add(Projections.property("numberOfFeet"))
            .add(Projections.property("description")));
```

  The sql query will be looked like :

```
select
    this_.id as y0_,
    this_.name as y1_,
    this_.food as y2_,
    this_.numberOfFeet as y3_,
    this_.description as y4_
from
    Animal this_
```

- **HCE Query**

  Code :

```
DetachedCriteria detachedCriteria = criteriaWithProjections(new String[]{
      "id", "name", "food", "numberOfFeet", "description"});
```

The sql query will be looked like :

```
select
    this_.id as y0_, this_.name
    as y1_, this_.food as y2_,
    this_.numberOfFeet as y3_,
    this_.description as y4_

from
    Animal this_
```

By using hibernate, we need to add projections one by one. HCE give an easy way to declare projections by only passing field name.

Case 4: Join Table (Association).

TODO: select id, name, country name of animal. (need join to Country)

- **Hibernate Criteria**

  Actually, when we select animal, it's country will be fetched automatically as the fetch type is EAGER. However, when we have a large of object and the projections declaration is needed for keeping performance. We must override the fetch type by using criteria, hql, or sql query. We will use criteria for the example.

  Code :

```
DetachedCriteria detachedCriteria =
      DetachedCriteria.forClass(Animal.class, "animal");
detachedCriteria.createAlias("animal.country", "country");
detachedCriteria.setProjection(Projections.projectionList()
      .add(Projections.property("animal.id"))
      .add(Projections.property("animal.name"))
      .add(Projections.property("country.name")) );
```

  Unfortunately, the code above throw an exception:
  java.lang.ClassCastException: [Ljava.lang.Object; cannot be cast to Animal.
  Hibernate cannot return an association selected field when using projections. In this case, we select "country.name" and hibernate cannot set the value of name to country. We need to declare new property and give alias for "country.name". This will happened to every associated projections.

- **HCE Query**

  Code :

```
DetachedCriteria detachedCriteria = criteriaWithProjections(new String[]
      {"id", "name", "country.name"});
```

  The sql query will be looked like :

```
select
    this_.id as y0_,
    this_.name as y1_,
    country0x1_.name as y2_
```

```
from
    Animal this_
inner join
    country country0x1_
        on this_.countryId=country0x1_.id
```

The example result will be looked like:

```
Animal@40e99ce5[id=1,name=Panda,numberOfFeet=0,food=<null>,description=<nu
ll>,country=softtech.hong.hce.junit.model.Country@9706da8[id=<null>,code=<
null>,name=China,description=<null>,shortname=<null>,active=false,cities=<
null>]]
```

Case 5: Select columns from one Table and add Equal Restrictions.

TODO: select id, name, and description from animal where name is Panda.

- **Hibernate Criteria**

Code :

```
DetachedCriteria detachedCriteria =
      DetachedCriteria.forClass(Animal.class, "animal");
            detachedCriteria.setProjection(Projections.projectionList()
                  .add(Projections.property("animal.id"), "id")
                  .add(Projections.property("animal.name"), "name")
                  .add(Projections.property("animal.description"),
                        "description")
                  );
            detachedCriteria.add(Restrictions.eq("animal.name",
                  "Panda"));
            detachedCriteria.setResultTransformer(new
                  AliasToBeanResultTransformer(Animal.class));
```

The sql query will be looked like :

```
select
    this_.id as y0_,
    this_.name as y1_,
    this_.description as y2_
from
    Animal this_
where
    this_.name=?
```

- **HCE Query**

Code :

```
DetachedCriteria detachedCriteria = criteriaByProperty(new String[]{"id",
"name", "description"}, Expression.eq("name", "Panda"));
```

The sql query will be looked like :

```
select
    this_.id as y0_,
    this_.name as y1_,
    this_.description as y2_
from
```

```
    Animal this_
where
    this_.name=?
```

Case 6: Select columns from first table and add Equal Restrictions to another.

TODO: select id, name, and description from animal where country name is China.

- **Hibernate Criteria**

  Code :

```
DetachedCriteria detachedCriteria =
      DetachedCriteria.forClass(Animal.class, "animal");
            detachedCriteria.createAlias("animal.country", "country");
            detachedCriteria.setProjection(Projections.projectionList()
                  .add(Projections.property("animal.id"), "id")
                  .add(Projections.property("animal.name"), "name")
                  .add(Projections.property("animal.description"),
                        "description")
                  );
            detachedCriteria.add(Restrictions.eq("country.name",
                  "China"));
            detachedCriteria.setResultTransformer(new
                  AliasToBeanResultTransformer(Animal.class));
```

  The sql query will be looked like :

```
select
    this_.id as y0_,
    this_.name as y1_,
    this_.description as y2_
from
    Animal this_
inner join
    country country1_
        on this_.countryId=country1_.id
where
    country1_.name=?
```

- **HCE Query**

  Code :

```
DetachedCriteria detachedCriteria = criteriaByProperty(new String[]{"id",
      "name", "description"}, Expression.eq("country.name", "China"));
```

  The sql query will be looked like :

```
select
    this_.id as y0_,
    this_.name as y1_,
    this_.description as y2_
from
    Animal this_
inner join
    country country0x1_
        on this_.countryId=country0x1_.id
```

```
    where
        country0x1_.name=?
```

We just declare “Expression.*eq*("country.name", "China")” for the restrictions. “country” is the field name for the association of Animal.

Case 7: Select columns from first table and second table, add Equal Restrictions to another.

TODO: select id, name, description, and country name from animal join country where country name is China.

- **Hibernate Criteria**

Code :

```
DetachedCriteria detachedCriteria =
      DetachedCriteria.forClass(Animal.class, "animal");
            detachedCriteria.createAlias("animal.country", "country");
            detachedCriteria.setProjection(Projections.projectionList()
                  .add(Projections.property("animal.id"), "id")
                  .add(Projections.property("animal.name"), "name")
                  .add(Projections.property("animal.description"),
                        "description")
                  .add(Projections.property("country.name"),
                        "countryName")
                  );
            detachedCriteria.add(Restrictions.eq("country.name",
                  "China"));
            detachedCriteria.setResultTransformer(new
                  AliasToBeanResultTransformer(Animal.class));
```

We need to add property **“countryName”** to Animal.java and mark it as @Transient. Hibernate projections cannot set country name value to country object in Animal class. Otherwise, it will throw exception :

```
org.hibernate.PropertyNotFoundException:
Could not find setter for countryName on
class softtech.hong.hce.junit.model.Animal
      at
org.hibernate.property.ChainedPropertyAccessor.getSetter(ChainedPropertyAccess
or.java:68)
      at
org.hibernate.transform.AliasToBeanResultTransformer.transformTuple(AliasToBea
nResultTransformer.java:87)
```

The sql query will be looked like

```
: select
    this_.id as y0_,
    this_.name as y1_,
    this_.description as y2_,
    country1_.name as y3_
from
    Animal this_
inner join
    country country1_
        on this_.countryId=country1_.id
```

```
    where
        country1_.name=?
```

- **HCE Query**

Code :

```
DetachedCriteria detachedCriteria = criteriaByProperty(new String[]{"id",
      "name", "description", "country.name" },
            Expression.eq("country.name", "China"));
```

The sql query will be looked like :

```
select
    this_.id as y0_,
    this_.name as y1_,
    this_.description as y2_,
    country0x1_.name as y3_
from
    Animal this_
inner join
    country country0x1_
        on this_.countryId=country0x1_.id
where
    country0x1_.name=?
```

Case 8: Restrictions Type (Like, Ilike, Greater Than, Less Than, IN, Empty, Not Empty, Null, Not Null, Or, and Between, etc).

TODO: select from animal where name [Restrictions Type] Panda.

- **Hibernate Criteria**

Code :

```
DetachedCriteria detachedCriteria =
      DetachedCriteria.forClass(Animal.class);

  Like :
     detachedCriteria.add(Restrictions.like("name", "Panda"));

  Ilike :
     detachedCriteria.add(Restrictions.ilike("name", "Panda"));
etc.
```

- **HCE Query**

Code :

```
  Like :
     DetachedCriteria detachedCriteria = criteriaByProperty(null,
           Expression.like("name", "Panda"));

  Ilike :
     DetachedCriteria detachedCriteria = criteriaByProperty(null, new
           Expression.ilike("name", "Panda"));
etc.
```

Case 9: Order

TODO: select from animal and order by name descending.

- **Hibernate Criteria**

  Code :

```
DetachedCriteria detachedCriteria =
        DetachedCriteria.forClass(Animal.class);
detachedCriteria.addOrder(Order.asc("name"));
```

- **HCE Query**

  Code :

```
DetachedCriteria detachedCriteria = criteriaWithProjections(null,
        softtech.hong.hce.model.Order.asc("name"));
```

  By passing null, we tell HCE to select all the field.

Case 10: Paging

TODO: select count from Animal, select data from Animal.

- **Hibernate Criteria**

  Code :

```
/* for count */
DetachedCriteria countCriteria = DetachedCriteria.forClass(Animal.class);
countCriteria.setProjection(Projections.rowCount());

/* for select */
DetachedCriteria selectCriteria =
        DetachedCriteria.forClass(Animal.class);
/* add restrictions here */
```

- **HCE Query**

  Code :

```
/* return array of DetachedCriteria, index 0 for count, index 1 for
        select */
DetachedCriteria[] detachedCriterias =
        queryTransformer(null, projections, orders);
```

  null mean that no filter is applied. See Case 11.

Case 11: Query By Example

TODO: select from Animal where animal object is as filter.

- **Hibernate Criteria**

  Code :

```
DetachedCriteria detachedCriteria =
        DetachedCriteria.forClass(Animal.class);
Animal animal = new Animal();
```

```
animal.setName("Panda");
detachedCriteria.add(Example.create(animal));
```

animal name will be automatically assigned as equal restriction.

- **HCE Query**

Code :

```
Animal animal = new Animal();
animal.setName("Panda");
DetachedCriteria detachedCriteria = queryTranslation(animal, null, null);
```

first null for projections, second one for ignore properties, in hibernate it's called exclude properties.

Case 11 : Query By Example on Association

TODO : select from Animal where animal object is as filter.

- **Hibernate Criteria**

Code :

```
DetachedCriteria detachedCriteria =
      DetachedCriteria.forClass(Animal.class);
Animal animal = new Animal();
animal.setName("Panda");
Country country = new Country();
country.setName("China");
animal.setCountry(country);
detachedCriteria.add(Example.create(animal));
```

only animal name will be automatically assigned as equal restriction. Country name will not be detected as restriction. QBE in hibernate cannot support an association.

- **HCE Query**

Code :

```
Animal animal = new Animal();
animal.setName( "Panda" );
Country country = new Country();
country.setName("China");
animal.setCountry(country);
DetachedCriteria detachedCriteria = queryTranslation(animal, null, null);
```

animal name and country name will be assigned as restrictions. HCE support QBE on association.

2.3. HCE Best Practice

HCE was built from hibernate criteria, the features inside using hibernate criteria and restrictions to build the query. HCE try to wrap the logic for building criteria one to the others.

The process of building criteria is based on hibernate configuration. HCE produce the DetachedCriteria class. We can also use hibernate Restrictions after getting DetachedCriteria from HCE. Here is the example:

TODO : select name from Animal, get the DetachedCriteria by using HCE then add Restrictions using hibernate "country name" is China

```
DetachedCriteria detachedCriteria = criteriaWithProjections(Animal.class, null);
detachedCriteria.add(Restrictions.eq("country0.name", "China" ));
```

You can see that first process is using HCE then we add restrictions equal using hibernate but something different, the alias of the country is "country0". Here is the point. HCE creating alias association by detecting the level's deep of association. The main class will be alias as "this". From the example above, if you want to add Restrictions using hibernate where animal id is 1, you should do :

```
detachedCriteria.add(Restrictions.eq("this.id", 1L ));
```

For a better understanding, let we add association from Animal to a new class, called as LifePlace class. Don't forget that Country has one to many association to City. Animal class will be:

```
@Entity
public class Animal {

   …

   @ManyToOne(fetch = FetchType.EAGER)
   @JoinColumn(name = "countryId", nullable = false, updatable = false) private Country
   country;

   @ManyToOne(fetch = FetchType.EAGER)
   @JoinColumn(name = "lifePlaceId", nullable = false, updatable = false) private
   LifePlace lifePlace;


      … // setter getter here

}
```

LifePlace class has id and name as attribute. Suppose that name value can be "water", "land", etc. Now, please imagine the association between Animal, LifePlace, Country, and City for while. Here is the description:

- Animal has foreign key to Country and LifePlace directly
- City has foreign key to Country directly
- Country one to many to City shown by cities attribute at Country class

From descriptions are shown above, HCE will alias the association or class join as following:

- Animal is a main class, the alias will be "this". To add Restrictions using hibernate to animal's attribute, the alias is "this". Name of Animal : "this.name"

- Country is an association to Animal, LifePlace is too. As Country and LifePlace can be directly access from Animal, hce will label them as level “0”. The alias of Country will be “country0” and LifePlace will be “lifePlace0” where country and lifePlace is field declaration at Animal class.
- To access City from Animal, we must join to Country. Using hce we can simply do “Expression.eq(“country.cities.name”, “Guang Dong”)”. If you want to use hibernate Restrictions, you can do “Restrictions.eq(“cities1.name”, “Guang Dong”)”. **One thing we must to remember is the city association must have been initiated, minimal you mention “country.cities.name” in the projections. Why?! To let hce process the association and know the level.**
- If the Country has association to Country itself. To reach Country of Country, the second country is the same level with city, just follow the one step above.

The example above is a simple case, it’s only an example, in the real case of course we will use hce Expression. Using hibernate Restrictions after getting DetachedCriteria from hce should be used if the logic or clause cannot be covered by hce. The other reason to use hibernate restrictions after hce process when we need to make choice, for example we only want to add restriction “country.name” if “x=1”, e.g:

```
DetachedCriteria detachedCriteria = criteriaWithProjections(Animal.class, null);
If(x == 1){
	detachedCriteria.add(Restrictions.eq(“country0.name”, “China” ));
}
```

The condition is checked outside hce. So, it depended of the logic business and code design.

One more thing, the Expression class of the hce can be manipulated. For the condition we mentioned above, hce can do it by using Expression. The expression can be used as following:

```
// if x = 1
Expression expression = new Expression();
If(x==1){
	expression.eq(“country.name”, “China”);
}
DetachedCriteria detachedCriteria = criteriaWithRestrictions(Animal.class,
	expression.getExpressions() );
```

It’s a choice, I think using hibernate after hce can be applied for exists, in. This version of hce has no exists and in feature.

Things to be concerned:

Multi level association one to many and many to many hasn’t been supported well. I suggest to let projections is null and just use the Expression. Multi level association mean: Country has many City and City has many District.

Conclusion

HCE is a extension library for hibernate, not a substitute library of hibernate, we can use it for simplifying query in criteria. Hibernate knowledge is needed for understanding HCE.

Finally, thanks to all readers and your suggestion is an improvement for me.

Hibernate Criteria Extension