

Hibernate Criteria



Lesson Objectives

1

- Understand the **Hibernate Object States/Lifecycle**.

2

- Understand the **HCQL** be used in Hibernate 4 or 5.

3

- Understand the **basic steps** to create a Criteria query.

4

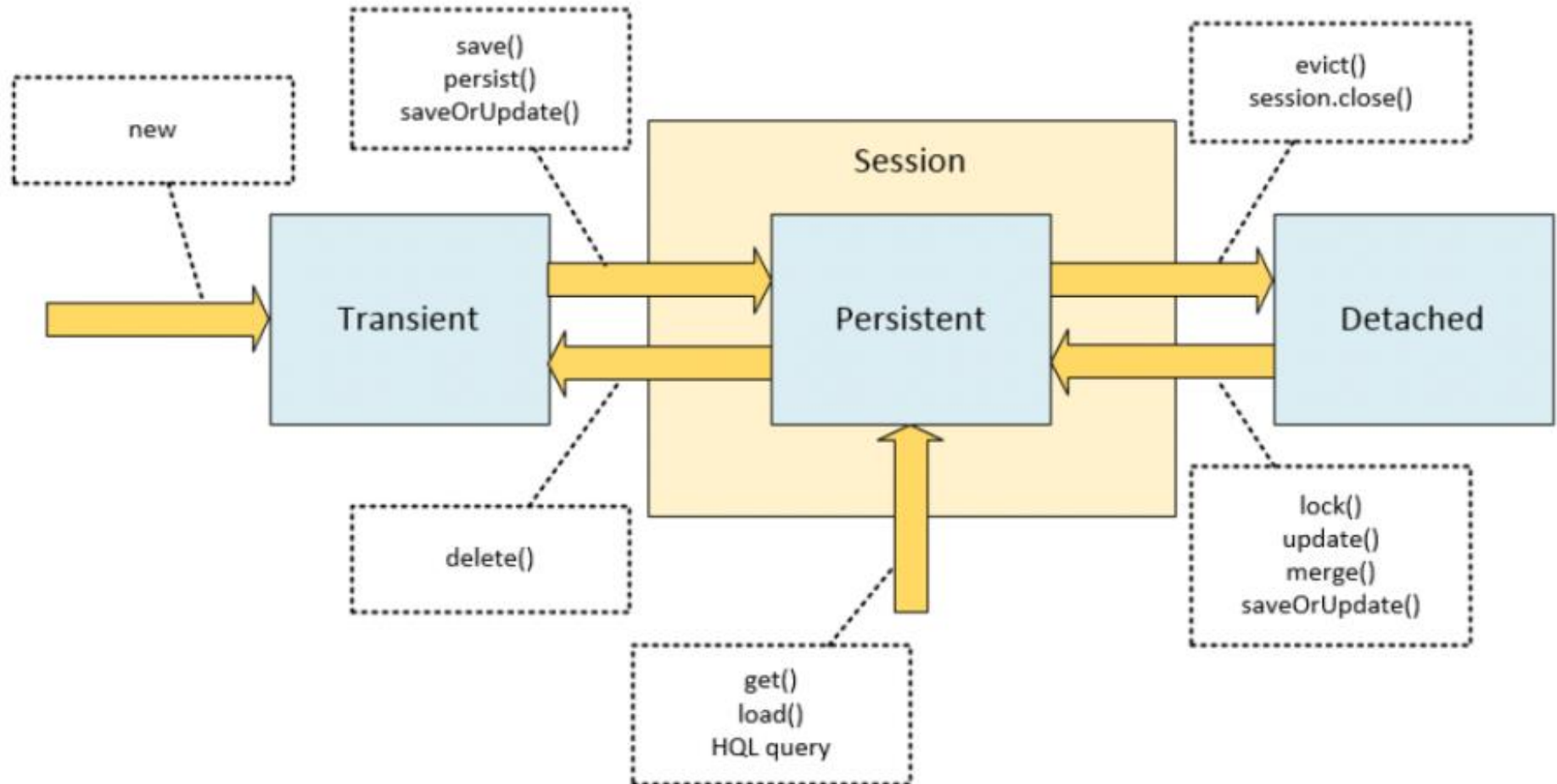
- Able to use Hibernate Query Language to **Join, Aggregation Functions, Pagination**.

- ❖ Hibernate Object States/Lifecycle
- ❖ Session methods: `save()`, `persist()`, `saveOrUpdate()`, `update()`, `merge()`
- ❖ Hibernate 5 Criteria Query Language
 - ✓ Introduction
 - ✓ Basic steps to create a CriteriaQuery
 - ✓ Hibernate Criteria Examples

Section 01

HIBETNATE OBJECT STATES/LIFECYCLE

Hibetnate Object States/Lifecycle



❖ Transient:

- ✓ The transient state is the **initial state of an object**.
- ✓ Once we create an instance of POJO (Plain Old Java Object) class, then the object entered in the transient state.
- ✓ An object is **not associated with the Session**. So, the transient state is **not related to any database**.
- ✓ The transient objects exist in the heap memory. They are independent of Hibernate.
- ✓ **Example:**

```
// Here, object enters in the transient state.  
1.Employee e=new Employee();  
2.e.setId(101);  
3.e.setFirstName("Gaurav");  
4.e.setLastName("Chawla");
```

❖ Persistent:

- ✓ As soon as the object **associated with the Session**, it entered in the persistent state.
- ✓ The object is in the **persistence state/persistence context** when we save or persist it.
- ✓ Each object represents the row of the database table.
- ✓ So, modifications in the data make changes in the database.
- ✓ **Example:**

```
1.session.save(e);  
2.session.persist(e);  
3.session.update(e);  
4.session.saveOrUpdate(e);  
5.session.lock(e);  
6.session.merge(e);
```

❖ Detached:

- ✓ Once we either **close the session** or **clear its cache**, then the object entered into the detached state.
- ✓ As an object is **no more associated with the Session**, modifications in the data don't affect any changes in the database.
- ✓ However, the detached object still has a representation in the database.
- ✓ If we want to persist the changes made to a detached object, it is required to reattach the application to a valid Hibernate session.
- ✓ To associate the detached object with the new hibernate session, use any of these methods - **lock()**, **merge()**, **refresh()**, **update()** or **save()** on a new session with the reference of the detached object.
- ✓ **Example:**

```
1.session.close();  
2.session.clear();  
3.session.detach(e);  
4.session.evict(e);
```


❖ Example:

```
Session session = factory.openSession();  
    Student student = new Student();  
    student.setName("chandrashekhar");  
Transaction transaction = session.beginTransaction();
```

} Student Object is in Transient State

```
    session.save(student);  
    transaction.commit();  
    session.close();
```

} Student Object is in Persistent State

student Detached state

- ❖ **void persist(Object o):** The *persist* method is intended for adding a new entity instance to the persistence context, i.e. transitioning an instance from transient to *persistent* state.

```
Person person = new Person();  
person.setName("John");  
session.persist(person);
```

- ✓ The *person* object has transitioned from *transient* to *persistent* state.
- ✓ The object is in the persistence context now, but **not yet saved to the database**.
- ✓ The generation of *INSERT* statements will occur only upon **committing the transaction, flushing or closing** the session.
- ✓ **If an instance is *detached*, you should expect an exception**, either upon calling this method, or upon committing or flushing the session.

- ❖ Serializable **save(Object o)**: The `save` method is an “original” Hibernate method that does not conform to the JPA specification.

```
Person person = new Person();  
person.setName("John");  
Long id = (Long) session.save(person);
```

- ✓ The call of `save` on a *detached* instance creates a **new persistent instance** and assigns it a new identifier, which results in a duplicate record in a database upon committing or flushing.

- ❖ **Object merge(Object o):** The main intention of the *merge* method is to update a *persistent* entity instance with new field values from a *detached* entity instance.

```
Person person = new Person();  
person.setName("John");  
session.save(person);  
  
session.evict(person);  
person.setName("Mary");  
  
Person mergedPerson = (Person) session.merge(person);
```

- ✓ If the entity is *detached*, it is copied upon an existing *persistent* entity;
- ✓ if the entity is *transient*, it is copied upon a newly created *persistent* entity;
- ✓ if the entity is *persistent*, then this method call does not have effect on it (but the cascading still takes place).

❖ **void update(Object o):** As with *persist* and *save*, the *update* method is an “original” Hibernate method that was present long before the *merge* method was added. Its semantics differs in several key points:

- the *update* method transitions the passed object from *detached* to *persistent* state;
- this method throws an exception if you pass it a *transient* entity.

```
Person person = new Person();  
person.setName("John");  
session.save(person);  
session.evict(person);  
  
person.setName("Mary");  
session.update(person);
```

- ❖ **void saveOrUpdate(Object o):** This method appears only in the Hibernate API and does not have its standardized counterpart. Similar to *update*, it also may be used for reattaching instances.

```
Person person = new Person();  
person.setName("John");  
session.saveOrUpdate(person);
```

- ✓ The main difference of *saveOrUpdate* method is that it **does not throw exception** when applied to a *transient* instance; instead, it makes this *transient* instance *persistent*.

- ❖ If you *don't have any special requirements*, as a rule of thumb, you should stick to the **persist** and **merge** methods, because they are standardized and guaranteed to conform to the JPA specification.
- ❖ They are also portable in case you decide to switch to *another persistence provider*, but they may sometimes appear not so useful as the “original” Hibernate methods, *save*, *update* and *saveOrUpdate*.

Section 02

HIBERNATE 5 CRITERIA QUERY LANGUAGE

- ❖ Criteria queries offer a **type-safe** alternative to HQL, JPQL and native SQL queries (*org.hibernate.Criteria* API).
- ❖ Criteria queries are a programmatic, type-safe way to express a query.
- ❖ Criteria queries are essentially an object graph, where each part of the graph represents an increasing (as we navigate down this graph) more atomic part of the query.

- ❖ The Criterion API (*org.hibernate.Criteria*) is one of the **best** parts of Hibernate.
- ❖ The syntax is **simple** and very **intuitive**, so it is **easy** for the java programmer to add criteria.
- ❖ It offers compile-time syntax checking, **convenience** of use.

The basic steps to create a Criteria query are:

- ❖ 1 - Create a CriteriaBuilder instance by calling the Session.getCriteriaBuilder() method.
CriteriaBuilder builder = session.getCriteriaBuilder();
- ❖ 2 - Create a query object by creating an instance of the CriteriaQuery interface.
CriteriaQuery<T> query = builder.createQuery(T.class);
- ❖ 3 - Set the query Root by calling the from() method on the CriteriaQuery object to define a range variable in FROM clause.
Root<T> root = query.from(T.class);
- ❖ 4 - Specify what the type of the query result will be by calling the select() method of the CriteriaQuery object.
query.select(root);
- ❖ 5 - Prepare the query for execution by creating a org.hibernate.query.Query instance by calling the Session.createQuery() method, specifying the type of the query result.
Query<T> q = session.createQuery(query);
- ❖ 6 - Execute the query by calling the getResultList() or getSingleResult() method on the org.hibernate.query.Query object.
List<T> list = q.getResultList();

❖ Example 1: Selecting an entity

```
CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery<Departments> criteria = builder
    .createQuery(Departments.class);

Root<Departments> root = criteria.from(Departments.class);
criteria.select(root);
List<Departments> departments = session.createQuery(criteria)
    .getResultList();
```

❖ Example 2: Selecting an expression

```
CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery<Departments> criteria = builder
    .createQuery(Departments.class);

Root<Departments> root = criteria.from(Departments.class);

criteria.multiselect(root.get("deptId"), root.get("deptName"));

List<Departments> departments = session.createQuery(criteria)
    .getResultList();
```

❖ Example 3: Selecting multiple values

```
CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery<Object[]> criteria = builder.
    createQuery( Object[].class );
Root<Person> root = criteria.from( Person.class );

Path<Long> idPath = root.get("id");
Path<String> nickNamePath = root.get("nickName");

criteria.select( builder.array( idPath, nickNamePath ) );
criteria.where( builder.
    equal(root.get("nickName"), "John Doe" ) );

List<Object[]> idAndNickNames = session.
    createQuery( criteria ).getResultList();
```

❖ Using *Expressions*

- ✓ The *CriteriaBuilder* can be used to restrict query results based on specific conditions.
- ✓ By using *CriteriaQuery where()* method and provide *Expressions* created by *CriteriaBuilder*.

❖ Common Examples:

- ✓ *To get items having a price more than 1000:*

```
criteria.select(root).where(builder.gt(root.get("itemPrice"), 1000));
```

- ✓ *Getting items having itemPrice less than 1000:*

```
criteria.select(root).where(builder.lt(root.get("itemPrice"), 1000));
```

- ✓ *Items having itemNames contain Chair:*

```
criteria.select(root).where(builder.like(root.get("itemName"), "%chair%"));
```

- ✓ *Records having itemPrice in between 100 and 200:*

```
criteria.select(root).where(builder.between(root.get("itemPrice"), 100, 200));
```

❖ Common Examples:

✓ *To check if the given property is null:*

```
criteria.select(root).where(builder.isNull(root.get("itemDescription")));
```

✓ *To check if the given property is not null:*

```
criteria.select(root).where(builder.isNotNull(root.get("itemDescription")));
```

❖ Criteria API allows us to easily chain expressions:

```
Predicate greaterThanPrice = builder  
    .gt(root.get("itemPrice"), 1000);  
Predicate chairItems = builder  
    .like(root.get("itemName"), "Chair%");  
criteria.where(builder.and(greaterThanPrice, chairItems));
```


❖ Sorting

```
criteria.orderBy(  
    builder.asc(root.get("itemName")),  
    builder.desc(root.get("itemPrice")));
```

❖ GROUP BY and HAVING example

```
List<Departments> departments = session.createQuery(criteria)
    .getResultList();

CriteriaBuilder builder = session.getCriteriaBuilder();

CriteriaQuery<Object[]> criteriaQuery =
    builder.createQuery(Object[].class);
Root<Employee> root = criteriaQuery.from(Employee.class);
criteriaQuery.multiselect(builder.count(root.get("name")),
    root.get("salary"), root.get("department"));
criteriaQuery.groupBy(root.get("salary"), root.get("department"));
criteriaQuery.having(builder.greaterThan(root.get("salary"), 30000));

Query<Object[]> query = session.createQuery(criteriaQuery);
List<Object[]> list = query.getResultList();
```

❖ FROM and JOIN example

```
session = HibernateUtils.getSessionFactory().openSession();
CriteriaBuilder builder = session.getCriteriaBuilder();
// Using FROM and JOIN
CriteriaQuery<Employees> criteriaQuery = builder
    .createQuery(Employees.class);
Root<Employees> empRoot = criteriaQuery.from(Employees.class);
Root<Departments> deptRoot = criteriaQuery.from(Departments.class);
criteriaQuery.select(empRoot);

criteriaQuery.where(builder.equal(empRoot.get("department"),
    deptRoot.get("deptId")));

Query<Employees> query = session.createQuery(criteriaQuery);
List<Employees> list = query.getResultList();

return list;
```

❖ HCQL Pagination

```
public List<UserInfor> search(int pageNumber, int pageSize) {  
    Session session = sessionFactory.getCurrentSession();  
    CriteriaBuilder criteriaBuilder = session.getCriteriaBuilder();  
    CriteriaQuery<UserInfor> criteriaQuery =  
        criteriaBuilder.createQuery(UserInfor.class);  
    Root<UserInfor> root = criteriaQuery.from(UserInfor.class);  
    criteriaQuery.select(root);  
  
    Query<UserInfor> query = session.createQuery(criteriaQuery);  
    query.setFirstResult((pageNumber - 1) * pageSize);  
    query.setMaxResults(pageSize);  
  
    List<UserInfor> listOfUser = query.getResultList();  
    sessionFactory.close();  
  
    return listOfUser;  
}
```

❖ Aggregate functions examples

✓ *Count number of employees:*

```
CriteriaQuery<Long> criteriaQuery = builder.createQuery(Long.class);  
Root<Employees> root = criteriaQuery.from(Employees.class);  
criteriaQuery.select(builder.count(root));  
Query<Long> query = session.createQuery(criteriaQuery);  
long count = query.getSingleResult();  
System.out.println("Count = " + count);
```

✓ *Get max salary*

```
CriteriaQuery<Integer> criteriaQuery = builder.createQuery(Integer.class);  
Root<Employees> root = criteriaQuery.from(Employees.class);  
criteriaQuery.select(builder.max(root.get("salary"))));  
Query<Integer> query = session.createQuery(criteriaQuery);  
int maxSalary = query.getSingleResult();  
System.out.println("Max Salary = " + maxSalary);
```

❖ Aggregate functions examples

✓ *Get Average Salary*

```
CriteriaQuery<Double> criteriaQuery = builder.createQuery(Double.class);  
Root<Employees> root = criteriaQuery.from(Employees.class);  
criteriaQuery.select(builder.avg(root.get("salary")));  
Query<Double> query = session.createQuery(criteriaQuery);  
double avgSalary = query.getSingleResult();  
System.out.println("Average Salary = " + avgSalary);
```

✓ *Count distinct employees*

```
CriteriaQuery<Long> criteriaQuery = builder.createQuery(Long.class);  
Root<Employees> root = criteriaQuery.from(Employees.class);  
criteriaQuery.select(builder.countDistinct(root));  
Query<Long> query = session.createQuery(criteriaQuery);  
long distinct = query.getSingleResult();  
System.out.println("Distinct count = " + distinct);
```

Section: tham khảo thêm

HIBERNATE 4 CRITERIA QUERY LANGUAGE

(CHỈ CẦN LỰA CHỌN HIBERNATE 4 OR 5)

```
public static List getStockDailyRecordCriteria(Date startDate, Date endDate,
    Long volume, Session session) {

    Criteria criteria = session.createCriteria(StockDailyRecord.class);
    if(startDate!=null) {
        criteria.add(Expression.ge("date", startDate));
    }
    if(endDate!=null) {
        criteria.add(Expression.le("date", endDate));
    }
    if(volume!=null) {
        criteria.add(Expression.ge("volume", volume));
    }
    criteria.addOrder(Order.asc("date"));

    return criteria.list();
}
```


The commonly used methods of Restrictions class are as follows:

- ❖ **public static SimpleExpression lt(String propertyName, Object value):** sets the **less than** constraint to the given property.
- ❖ **public static SimpleExpression le(String propertyName, Object value):** sets the **less than or equal** constraint to the given property.
- ❖ **public static SimpleExpression gt(String propertyName, Object value):** sets the **greater than** constraint to the given property.
- ❖ **public static SimpleExpression ge(String propertyName, Object value):** sets the **greater than or equal** than constraint to the given property.
- ❖ **public static SimpleExpression ne(String propertyName, Object value):** sets the **not equal** constraint to the given property.
- ❖ **public static SimpleExpression eq(String propertyName, Object value):** sets the **equal** constraint to the given property.
- ❖ **public static Criterion between(String propertyName, Object low, Object high):** sets the **between** constraint.
- ❖ **public static SimpleExpression like(String propertyName, Object value):** sets the **like** constraint to the given property.

❖ Restrictions.eq, lt, le, gt, ge

*(Criteria Queries : Equal (**eq**), Not Equal(**ne**), Less than (**lt**), Less than or equal(**le**), greater than (**gt**), greater than or equal(**ge**) and Ordering the results(**Order.asc/desc**))*

Make sure the volume is great than 10000.

```
Criteria criteria = session.createCriteria(StockDailyRecord.class)
    .add(Restrictions.gt("volume", 10000));
```

❖ Restrictions.like, between, isNull, isNotNull

Make sure the stock name is start with 'MKYONG' and follow by any characters.

```
Criteria criteria = session.createCriteria(StockDailyRecord.class)
    .add(Restrictions.like("stockName", "MKYONG%"));
```

(<http://docs.jboss.org/hibernate/core/3.3/api/org/hibernate/criterion/Expression.html>)

❖ Criteria ordering query

```
criteria.addOrder(Order.asc("dateOfBirth"));  
criteria.addOrder(Order.desc("salary"));
```

❖ Criteria restrictions query

```
// HQL: tr.storeId=:storeId  
criteria.add(Restrictions.eq("tr.storeId", storeId));  
// HQL: tr.r.trafficTime>=:minDate  
criteria.add(Restrictions.ge("tr.trafficTime", minDate));  
// HQL: tr.r.trafficTime<:maxDate  
criteria.add(Restrictions.lt("tr.trafficTime", maxDate));  
criteria.add(Restrictions.between("dateOfBirth",  
                                startDate, endDate));
```

❖ Criteria restrictions query

```
criteria.add(Restrictions.like("name", "%th%"));
```

```
criteria.add(Restrictions.like("name", "%" + name + "%"));
```

Projections & Aggregations

```
Criteria cr = session.createCriteria(Employee.class);
```

// To get total row count.

```
cr.setProjection(Projections.rowCount());
```

// To get average of a property.

```
cr.setProjection(Projections.avg("salary"));
```

// To get distinct count of a property.

```
cr.setProjection(Projections.countDistinct("firstName"));
```

// To get maximum of a property.

```
cr.setProjection(Projections.max("salary"));
```

// To get minimum of a property.

```
cr.setProjection(Projections.min("salary"));
```

// To get sum of a property.

```
cr.setProjection(Projections.sum("salary"));
```

Create Criteria in Hibernate 4

```
public List<UserInfor> search(int pageNumber, int pageSize,
                             int searchType, int departmentId) {

    Session session = sessionFactory.openSession();
    Criteria criteria = session.createCriteria(UserInfor.class);
    // Criteria with @JoinColumn (User (N) - (1) Department
    Criteria depCrit = criteria.createCriteria("department");
    if(departmentId > 0) {
        depCrit.add(Restrictions.like("departmentId", departmentId ));
    }
    if (searchType == 1) {
        criteria.add(Restrictions.like("isAdmin", 1));
    }
    if (searchType == 2) {
        criteria.add(Restrictions.like("isEnabled", 0));
    }
    if (searchType == 3) {
        criteria.add(Restrictions.like("isEnabled", 1));
    }
    criteria.setFirstResult((pageNumber - 1) * pageSize);
    criteria.setMaxResults(pageSize);

    List<UserInfor> listOfUser = criteria.list();

    return listOfUser;
}
```

- ❖ Hibernate Object States/Lifecycle
- ❖ Session methods: `save()`, `persist()`, `saveOrUpdate()`, `update()`, `merge()`
- ❖ Hibernate 5 Criteria Query Language
 - ✓ Introduction
 - ✓ Basic steps to create a CriteriaQuery
 - ✓ Hibernate Criteria Examples

Thank you

