Hibernate Query

Hibernate - links

- 1. https://docs.jboss.org/hibernate/orm/4.1/manual/en-US/html/ch16.html
- 2.https://www.tutorialspoint.com/hibernate/hibernate_query_language.html
- 3. https://www.java2novice.com/hibernate/session-interface-methods/
- 3. https://www.baeldung.com/jpa-join-types
- 4. https://docs.jboss.org/hibernate/orm/3.2/api/org/hibernate/Query.html
- 5. https://dzone.com/articles/hibernate-query-language
- 6. https://examples.javacodegeeks.com/enterprise-java/hibernate/hibernate-crud-operations-tutorial/
- 7. https://www.java4s.com/hibernate/

Types

- HQL
- Criteria Queries
- Native SQL

Hibernate Query Language

- Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database and it is Object Oriented SQL.
- Instead of table name, we use class name in HQL. So it is database independent query language.
- There are many advantages of HQL. They are as follows:
  - database independent
  - supports polymorphic queries
  - easy to learn for Java Programmer

Query Interface

- It is an object oriented representation of Hibernate Query.
- The object of Query can be obtained by calling the createQuery() method Session interface.
- The query interface provides many methods. There is given commonly used methods:
  - **public int executeUpdate()** is used to execute the update or delete query.
  - **public List list()** returns the result of the ralation as a list.
  - **public Query setFirstResult(int rowno)** specifies the row number from where record will be retrieved.
  - **public Query setMaxResult(int rowno)** specifies the no. of records to be retrieved from the relation (table).
  - **public Query setParameter(int position, Object value)** it sets the value to the JDBC style query parameter.
  - **public Query setParameter(String name, Object value)** it sets the value to a named query parameter.
  - uniqueResult()

HQL SELECT Statement

```
[SELECT [DISTINCT] property [, ...]]
   FROM path [[AS] alias] [, ...] [FETCH ALL PROPERTIES]
   WHERE logicalExpression
   GROUP BY property [, ...]
   HAVING logicalExpression
   ORDER BY property [ASC | DESC] [, ...]
```

HQL - FROM

- You will use **FROM** clause if you want to load a complete persistent objects into memory

```
String hql = "FROM Employee";
Query query = session.createQuery(hql);
List results = query.list();
```

- If you need to fully qualify a class name in HQL, just specify the package and class name

```
String hql = "FROM com.hibernatebook.criteria.Employee";
Query query = session.createQuery(hql);
List results = query.list();
```

HQL - AS

- The **AS** clause can be used to assign aliases to the classes in your HQL queries.

```
String hql = "FROM Employee AS E";
Query query = session.createQuery(hql);
List results = query.list();
```

- The **AS** keyword is optional and you can also specify the alias directly after the class name

```
String hql = "FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

HQL - SELECT Clause

- The **SELECT** clause provides more control over the result set then the from clause.
- If you want to obtain few properties of objects instead of the complete object, use the SELECT clause

```
String hql = "SELECT E.firstName FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

HQL - WHERE Clause

- If you want to narrow the specific objects that are returned from storage, you use the WHERE clause

```
String hql = "FROM Employee E WHERE E.id = 10";
Query query = session.createQuery(hql);
List results = query.list();
```

-
  ```
  select cat.name from DomesticCat cat
  where cat.name like 'fri%'
  ```

HQL – unique result

```
String sql = "FROM Task where id =:id";
Query query = session.createQuery(sql);
query.setParameter("id", n);

Task task = (Task) query.getSingleResult();
```

HQL - Named Parameters

- Hibernate supports named parameters in its HQL queries.
- This makes writing HQL queries that accept input from the user easy and you do not have to defend against SQL injection attacks.

- Pass an unchecked value from user input to the database will raise security concern, because it can easy get hack by SQL injection

  String hql = “From Stock s where s.stockCode = '" + stockCode + "'"; List result = session.createQuery(hql).list(); // risky code

HQL - Named Parameters

- The **setParameter** is smart enough to discover the parameter data type for you.

```
String hql = "FROM Employee E WHERE E.id = :employee_id";
Query query = session.createQuery(hql);
query.setParameter("employee_id",10);
List results = query.list();


Query query = session.createQuery("From ContactEntity Where firstName = :paramName");
query.setParameter("paramName", "Nick");
List list = query.list();
```

HQL - Named Parameters

- **setString** to tell Hibernate this parameter date type is String.

```
String hql = “From Stock s Where  s.stockCode = :stockCode";
List result = session.createQuery(hql).setString("stockCode", "7277").list();
```

-

HQL – Positional parameters

- It’s use question mark (?) to define a named parameter, and you have to set your parameter according to the position sequence.

```
String hql = “From Stock s Where s.stockCode = ?0 and s.stockName = ?1";
List result = session.createQuery(hql)
  . setParameter(0, "7277")
  .setParameter(1, "DIALOG")
  .list();
```

HQL - ORDER BY Clause

```
String hql = "FROM Employee E WHERE E.id > 10 ORDER BY E.salary DESC";
Query query = session.createQuery(hql);
List results = query.list();
```

```
String hql = "FROM Employee E WHERE  E.id > 10 " + "ORDER BY E.firstName  DESC, E.salary DESC ";
Query query = session.createQuery(hql);
List results = query.list();
```

HQL - GROUP BY Clause

```
String hql = "SELECT SUM(E.salary), E.firtName FROM Employee E " +"GROUP BY E.firstName";
Query query = session.createQuery(hql);
List results = query.list();
```

HQL - **Aggregate functions**

- The supported aggregate functions are:

  avg(...), sum(...), min(...), max(...)

  count(*)

  count(...), count(distinct ...), count(all...)

```
String hql = "SELECT  count(distinct E.firstName) FROM Employee E";
Query query = session.createQuery(hql);
List results = query.list();
```

HQL - UPDATE Clause

```
String hql = "UPDATE Employee set salary = :salary "  + "WHERE id = :employee_id";
Query query = session.createQuery(hql);

query.setParameter("salary", 1000);
query.setParameter("employee_id", 10);

int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

HQL - DELETE Clause

```
String hql = "DELETE  FROM Employee  WHERE id = :employee_id";

Query query = session.createQuery(hql);
query.setParameter("employee_id", 10);

int result = query.executeUpdate();
System.out.println("Rows affected: " + result);
```

HQL - INSERT Clause

- HQL supports **INSERT INTO** clause only where records can be inserted from one object to another object.

```
String hql = "INSERT INTO Employee(firstName, lastName, salary)"  +
                        "SELECT firstName, lastName, salary FROM old_employee";

Query query = session.createQuery(hql);
int result = query.executeUpdate();

System.out.println("Rows affected: " + result);
```

HQL - JOIN

```
public class Employee {
    @ManyToOne
    private Department department;

}


public class Department {

      @OneToMany(mappedBy = "department")
      private List<Employee> employees;

}



 TypedQuery<Department> query

      = entityManager.createQuery(

          "SELECT d FROM Employee e JOIN e.department d", Department.class);

    List<Department> resultList = query.getResultList();
```

HQL – JOIN (result not entity class)

```
public class Customer {
  // customerId, customerName, customerCity

   @OneToMany

   @JoinColumn(name = "cid",referencedColumnName="cid")

   private List items;

}


public class Item {

// itemId, itemName, price;

}
```

HQL - JOIN

```
Query qry= session.createQuery("Select c.customerName, c.customerCity,
                    i.itemName,i.price From Customer c  right join c.items i");

List l = qry.list();
Iterator it=l.iterator();

while(it.hasNext())

{

      Object rows[] = (Object[])it.next();

      System.out.println(rows[0]+ " -- " +rows[1] + "--"+rows[2]+"--"+rows[3]);

}
```