Hibernate Queries

Design by: DieuNT1

Lesson Objectives

1
- Understand the queries be used in hibernate.

2
- Understand the Native Query and @NamedNativeQuery.

3
- Able to use Hibernate Query Language.

4
- Understand the Proxy Object in Hibernate.

5
- Able to distinguish get() and load() method.

Agenda

- Queries Introduction
- Native Query
- Hibernate Query Language
- Hibernate Named Query
- Proxy Object
- get() vs load() method

Section 01

QUERIES INTRODUCTION

Hibernate Query Language

- The Hibernate Query Language (HQL) and Java Persistence Query Language (JPQL) are both object model focused query languages similar in nature to SQL.
  - ✓ JPQL is a heavily-inspired-by subset of HQL.
  - ✓ A JPQL query is always a valid HQL query, however the reverse is not true.
  - ✓ Both HQL and JPQL are non-type-safe ways to perform query operations. Criteria queries offer a type-safe approach to querying.

Navite Query

- You may also express queries in the native SQL dialect of your database.
  - ✓ This is useful if you want to *utilize database specific features* such as query hints or the CONNECT BY option in Oracle.
  - ✓ It also provides a clean migration path from a direct SQL/JDBC based application to Hibernate.
  - ✓ Note that Hibernate allows you to specify handwritten SQL (including stored procedures) for all **create**, **update**, **delete**, and **load** operations.

Hibernate Native Query

Hibernate Named Query

- A named query is a ***JPQL*** or ***Navite SQL*** expression with a predefined unchangeable query string.
  - ✓ You can define a named query either in *hibernate mapping file* or in *an entity class*.
  - ✓ The named queries in hibernate is a technique to group the HQL statements in a single location, and lately refer them by some name whenever the need to use them.
  - ✓ It helps largely in code cleanup because these HQL statements are no longer scattered in whole code.

Section 02

NATIVE QUERY

Native Query

- Hibernate allows us to execute the native SQL queries for all *create, update, delete* and *retrieve* operations.
- In hibernate, you can execute your native SQL queries using the **Session.createNativeQuery()** method..
  - ✓ Hibernate SQL query is not the recommended approach because we loose benefits related to hibernate association and *hibernate first level cache*.
- Query object:
  - ✓ **Syntax** to create the Query object and execute it:

```
Query<Employees> query = session.createNativeQuery(String query);
```

  - ✓ **SQLQuery Methods:**
    - **List<Object>** list() method: returns the list of Object array, we need to explicitly parse them to double, long etc.
    - addEntity() and addJoin() methods to fetch the data from associated table using tables join

Native Query

- **Example 1**: using addEntity()

```
@Override
   public List<Jobs> findAll() throws Exception {

         Session session = null;

         try {
             session = HibernateUtils.getSessionFactory().openSession();
             Query<Jobs> query = session
                     .createNativeQuery("SELECT * FROM dbo.Jobs")
                     .addEntity(Jobs.class);

             return query.list();
         } finally {
             if (session != null) {
                 session.close();
             }
         }
   }
```

- **Results:**

```
[Jobs [jobId=J01, jobTitle=Java Dev1, minSalary=1000.0, maxSalary=2000.0],
Jobs [jobId=J02, jobTitle=Java Dev2, minSalary=1200.0, maxSalary=2200.0],
Jobs [jobId=J03, jobTitle=Java Dev3, minSalary=1400.0, maxSalary=3200.0]]
```

Native Query

- **Example 2**: native query with the conditions/parameters

```
@Override
    public List<Jobs> findByNameAndSalary(String title, double salary)
            throws Exception {
        Session session = null;

        try {
            session = HibernateUtils.getSessionFactory().openSession();

            Query query = session.createNativeQuery(
                    "SELECT * FROM dbo.Jobs j WHERE j.job_title LIKE :title "
                    + "AND j.min_salary <= :salary AND j.max_salary >= :salary")
                    .addEntity(Jobs.class);

            query.setParameter("title", "%" + title + "%");
            query.setParameter("salary", salary);

            return query.list();

        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
```

- **Results:**

```
[Jobs [jobId=J01, jobTitle=Java Dev1, minSalary=1000.0, maxSalary=2000.0]]
```

Native Query

❖ **Example 3**: addEntity(), addJoin()

```
@Override
    public List<Object[]> findAll() throws Exception {
        Session session = null;
        try {
            session = HibernateUtils.getSessionFactory().openSession();
            Query query = session.createNativeQuery(
                    "SELECT j.*, e.* FROM dbo.Jobs j JOIN dbo.Employees e "
                    + "ON j.job_id = e.job_id")
                      .addEntity("j", Jobs.class)
                      .addJoin("e", "j.employees");

            List<Object[]> jobs = query.list();

            return jobs;

        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
```

Native Query

```java
@Test
void testFindAll() throws Exception {
	List<Object[]> jobs = jobDao.findAll();

	for (Object[] object : jobs) {
		Jobs job = (Jobs) object[0];
		System.out.println(job);

		for (Employees employee : job.getEmployees()) {
		    System.out.println(employee);
		}
	}
}
```

Results:

```
Jobs [jobId=J01, jobTitle=Java Dev1, minSalary=1000.0, maxSalary=2000.0]
Employees     [employeeId=5, first_name=Nguyen, last_name=Minh Thanh, email=thanh@fsoft.com.vn, phoneNumber=0988777111,
              hireDate=1999-01-01, salary=1000.0, commissionPct=1.1]
Employees     [employeeId=1, first_name=Nguyen, last_name=Quang Anh, email=anhnd22@fsoft.com.vn, phoneNumber=0988777666,
              hireDate=2019-01-01, salary=1000.0, commissionPct=1.1]
Jobs [jobId=J01, jobTitle=Java Dev1, minSalary=1000.0, maxSalary=2000.0]
Employees     [employeeId=5, first_name=Nguyen, last_name=Minh Thanh, email=thanh@fsoft.com.vn, phoneNumber=0988777111,
              hireDate=1999-01-01, salary=1000.0, commissionPct=1.1]
Employees     [employeeId=1, first_name=Nguyen, last_name=Quang Anh, email=anhnd22@fsoft.com.vn, phoneNumber=0988777666,
              hireDate=2019-01-01, salary=1000.0, commissionPct=1.1]
Jobs [jobId=J02, jobTitle=Java Dev2, minSalary=1200.0, maxSalary=2200.0]
Employees     [employeeId=7, first_name=Hoang, last_name=Van Liem, email=Liem@fsoft.com.vn, phoneNumber=0988777112,
              hireDate=1999-01-01, salary=1000.0, commissionPct=1.1]
```

Native Query

- Using **@NamedNativeQuery** and **@NamedNativeQueries** Annotations.
- **Syntax**:

```
@Entity
@Table(name = "Employees", schema = "dbo", indexes = {
        @Index(columnList = "first_name, last_name", name = "IDX_EMP_NAME") })
@NamedNativeQueries({
         @NamedNativeQuery(name = "FIND_EMP_BY_JOB", query = "SELECT e.* "
                + "FROM dbo.Employees e JOIN dbo.Jobs j ON e.job_id = j.job_id "
                + "AND j.job_id LIKE :jobTitle", resultClass = Employees.class),
         @NamedNativeQuery(name = "EMP_FIND_ALL",
                       query = "SELECT * FROM dbo.Employees",
                       resultClass = Employees.class)
         @NamedNativeQuery(name = "COUNT_EMP",
              query = "SELECT AVG(e.salary) FROM dbo.Employees e "
              + "WHERE e.job_id = :jobId")})
public class Employees {
}
```

Native Query

- The session.createNamedQuery(String name) method:

```
@Override
   public List<Employees> findByJob(String jobTile) {
        Session session = null;

        try {
             session = HibernateUtils.getSessionFactory().openSession();

             Query<Employees> query = session
                     .createNamedQuery("FIND_EMP_BY_JOB");

             query.setParameter("jobTitle", "%" + jobTile + "%");
             return query.list();

        } finally {
             if (session != null) {
                  session.close();
             }
        }
   }
```

Native Query

❖ The query.getSingleResult() method:

```
@Override
    public double countByJob(String jobId) {
         Session session = null;

         try {
              session = HibernateUtils.getSessionFactory().openSession();

              Query query = session.createNamedQuery("COUNT_EMP");
              query.setParameter("jobId", jobId);

              return (double) query.getSingleResult();

         } finally {
              if (session != null) {
                   session.close();
              }
         }
    }
```

Section 03

HIBERNATE QUERY LANGUAGE

Hibernate Query Language (HQL)

- Syntax is quite similar to database SQL language.
- Uses class name instead of table name, and property names instead of column name:
  - ✓ **SQL similarity:** HQL's syntax is very similar to standard SQL.
  - ✓ **Fully object-oriented:** HQL doesn't use real names of table and columns. It uses class and property names instead. HQL can understand inheritance, polymorphism and association.
  - ✓ **Case-insensitive for keywords:** Like SQL, keywords in HQL are case-insensitive. That means SELECT, select or Select are the same.
  - ✓ **Case-sensitive for Java classes and properties:** HQL considers case-sensitive names for Java classes and their properties, meaning Person and person are two different objects.

Execute HQL in Hibernate

- Write your HQL:

```
String hql = "FROM Projects WHERE startDate >= :startDate";
```

- Create a Query from the Session:

```
Query query = session.createQuery(hql);
```

- Set parameter (if need):

```
query.setParameter("startDate", startDate);
```

- Execute the query: depending on the type of the query (listing or update), an appropriate method is used:
  - ✓ **For a listing query (SELECT):**

```
List listResult = query.list();
```

  - ✓ **For an update query (INSERT, UPDATE, DELETE):**

```
int rowsAffected = query.executeUpdate();
```

Execute HQL in Hibernate

- Extract result returned from the query: depending of the type of the query, Hibernate returns different type of result set.
  - ✓ Select query on a mapped object returns a list of those objects.
  - ✓ Join query returns a list of arrays of Objects which are aggregate of columns of the joined tables. This also applies for queries using aggregate functions (count, sum, avg, etc).
- **Join Query,** HQL supports the following join types (similar to SQL):
  - ✓ INNER JOIN (can be abbreviated as JOIN).
  - ✓ LEFT OUTER JOIN (can be abbreviated as LEFT JOIN).
  - ✓ RIGHT OUTER JOIN (can be abbreviated as RIGHT JOIN).
  - ✓ FULL JOIN

Execute HQL in Hibernate

❖ **Example 1**: Join query

```
public List<Object[]> findPublisherBook() {
        Session session = null;
        try {
            session = HibernateUtils.getSessionFactory().openSession();

            String joinQuery = "FROM Publisher p INNER JOIN p.publisherBook pb";

            Query quey = session.createQuery(joinQuery);

            return quey.list();

        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
```

Execute HQL in Hibernate

- **Example 1**: Join query

```
    @Test
        void testFindPublisherBook() {
            List<Object[]> objects = publisherDao.findPublisherBook();

            for (Object[] object : objects) {
                System.out.println((Publisher) object[0]);

                System.out.println((PublisherBook) object[1]);
            }

        }
```

- **Results:**

```
Publisher [publisherId=1, name=NXB GD, phone=0979867234]

PublisherBook [id=PublisherBookId [publisherId=1, bookId=1], format=ABC]
```

Execute HQL in Hibernate

❖ **Example 2**: Join query

```
public List<Object[]> findPublisherBook() {
        Session session = null;
        try {
            session = HibernateUtils.getSessionFactory().openSession();

            String joinQuery = "FROM Publisher p JOIN p.publisherBook pb "
                        + "JOIN pb.book b ";
                        // The same as
                        // "FROM Publisher p INNER JOIN p.publisherBook pb "
                        // + "ON p.publisherId = pb.publisher.publisherId "
                        // + "INNER JOIN Book b "
                        // + "ON b.bookId = pb.book.bookId";

            Query quey = session.createQuery(joinQuery);

            return quey.list();

        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
```

Execute HQL in Hibernate

- **Example 2**: Join query
- **Results:**

```
Publisher [publisherId=1, name=NXB GD, phone=0979867234]

PublisherBook [id=PublisherBookId [publisherId=1, bookId=1], format=ABC]

Book [bookId=1, title=Java SE, year=2020, version=1.0]
```

Hibernate Query Language

- **Example 2:**
  - Update a stock name to “DIALOG1” where stock code is “7277”

```
Query query = session.createQuery("update Stock set stockName = :stockName" +
                                  " where stockCode = :stockCode");
query.setParameter("stockName", "DIALOG1");
query.setParameter("stockCode", "7277");
int result = query.executeUpdate();
```

  - Delete a stock where stock code is “7277”

```
Query query = session.createQuery("delete Stock where stockCode = :stockCode");
query.setParameter("stockCode", "7277");
int result = query.executeUpdate();
```

Hibernate Query Language

❖ Example 3: Sort Query

```
@Override
    public List<Projects> searching(LocalDate startDate) throws Exception {
        Session session = null;

        try {
            session = HibernateUtils.getSessionFactory().openSession();

            String hql = "FROM Projects WHERE startDate >= :startDate “
                    + "ORDER BY completedOn DESC";

            Query<Projects> query = session.createQuery(hql);

            query.setParameter("startDate", startDate);

            return query.list();

        } finally {
            if (session != null) {
                session.close();
            }
        }
    }
```

Hibernate Query Language

❖ **Example 4: Group By**

```
String hql = "SELECT SUM(p.price), p.category.name "
           + "FROM Product p GROUP BY category";

Query query = session.createQuery(hql);
List<Object[]> listResult = query.list();

for (Object[] aRow : listResult) {
        Double sum = (Double) aRow[0];
        String category = (String) aRow[1];
        System.out.println(category + " - " + sum);
}
```

Hibernate Query Language

- **Example 5**: Pagination Query

  ✓ To return a subset of a result set, the Query interface has two methods for limiting the result set:

    - setFirstResult(intfirstResult): sets the first row to retrieve.
    - setMaxResults(intmaxResults): sets the maximum number of rows to retrieve.

```
Query query = session.createQuery("FROM Employees");

query.setFirstResult(0);
query.setMaxResults(10);

return query.list();
```

Hibernate Query Language

❖ **Example 6**: Using Aggregate Functions

HQL supports the following aggregate functions:

- ✓ avg(…), sum(…), min(…), max(…)
- ✓ count(*)
- ✓ count(…), count(distinct…), count(all…)

```
String hql = "SELECT COUNT(jobTitle) FROM Jobs";


Query query = session.createQuery(hql);
List listResult = query.list();
Number number = (Number) listResult.get(0);
System.out.println(number.intValue());
```

Named Query

- The hibernate named query is way to use any query by some meaningful name. It is like using alias names.

- So that application programmer need not to **scatter** queries to all the java code.

- There are two ways to define the named query in hibernate:
  - by annotation
  - by mapping file

Named Query (cont)

- Named Query by Annotation:
  - **@NameQueries:** is used to define the multiple named queries.
  - **@NameQuery:** is used to define the single named query.

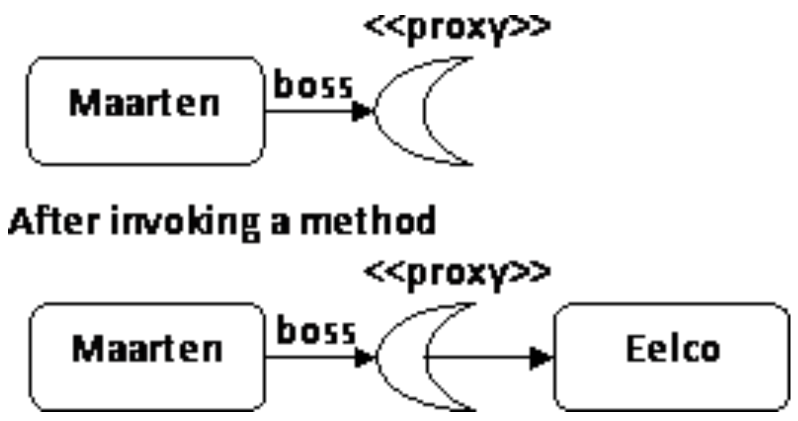
```
@NamedQueries(
   {
      @NamedQuery(
      name = "findEmployeeByName",
      query = "from Employee e where e.name = :name"
      )
   }
)
```

Section 04

PROXY OBJECT

Proxy Object

- An object proxy is just a way to avoid retrieving an object until you need it.
- The Proxy class is generated at runtime and it extends the original entity class.
- Uses Proxy objects for entities is for to allow lazy loading.
- When accessing basic properties on the Proxy, it simply delegates the call to the original entity.



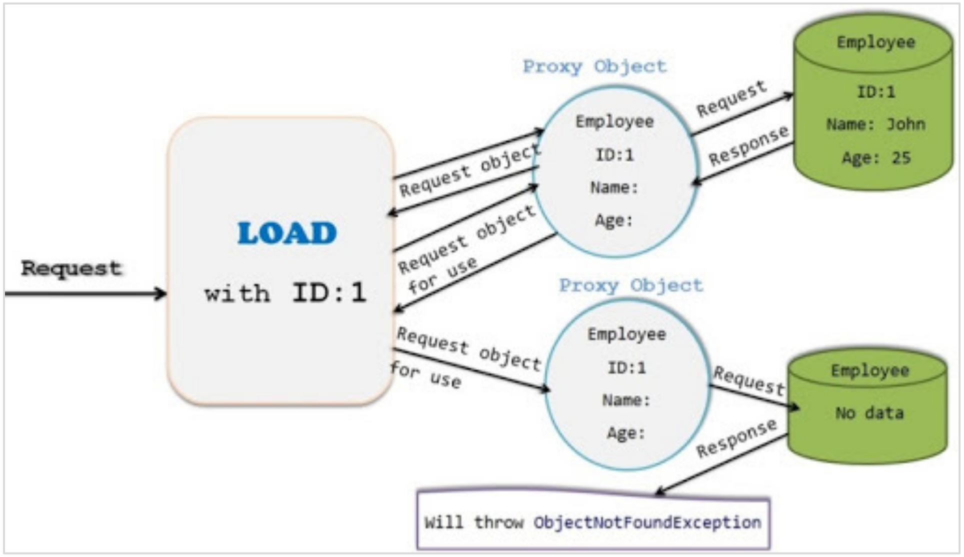
get() and load() Method

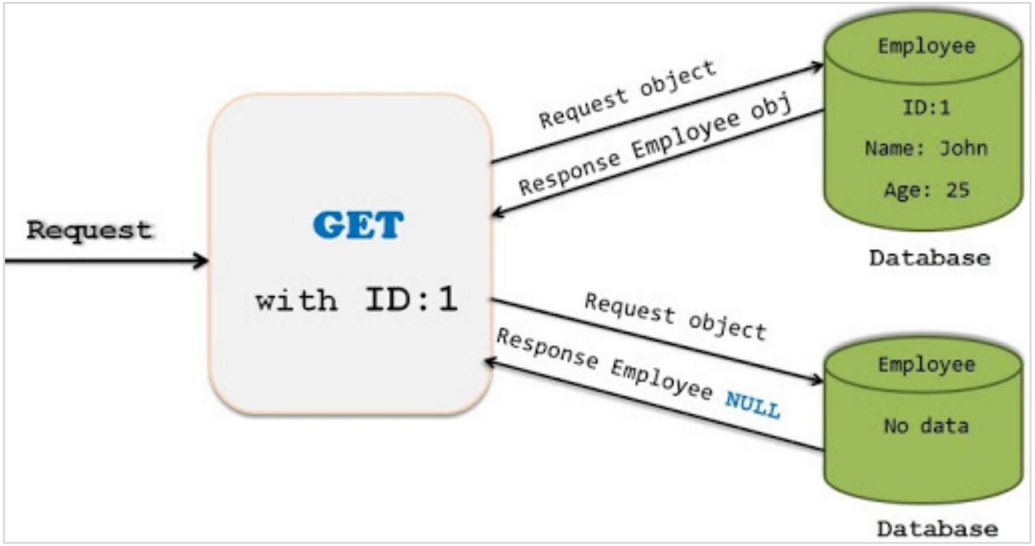
- In hibernate, get() and load() are two methods which is used to fetch data for the given identifier.
- They both belong to Hibernate session class.
- get() method return null: If no row is available in the session cache or the database for the given identifier
- load() method throws object not found exception.

```
// Get Example
User user = (User) session.get(User.class, new
Integer(2));


// Load Example
User user = (User) session.load(User.class, new
Integer(2));
```

get() and load() Method





Get and Load Method

❖ Difference between get() and load()

Key	get()	load()
Basic	It is used to fetch data from the database for the given identifier	It is also used to fetch data from the database for the given identifier
Null Object	It object not found for the given identifier then it will return null object	It will throw object not found exception
Lazy or Eager loading	It returns fully initialized object so this method eager load the object	It always returns proxy object so this method is lazy load the object
Performance	It is slower than load() because it return fully initialized object which impact the performance of the application	It is slightly faster.
Use Case	If you are not sure that object exist then use get() method	If you are sure that object exist then use load() method

Summary

- Queries Introduction
- Native Query
- Hibernate Query Language
- Hibernate Named Query
- Proxy Object
- get() vs load() method

Thank you