2016

Hibernate Material

1. What is hibernate?

Hibernate: hibernate-core: 5.1 (Latest Version)

Hibernate is an open-source and lightweight ORM tool that is used to store, manipulate and retrieve data from the database. Hibernate is an Object-Relational Mapping (ORM) solution for JAVA and it raised as an open source persistent framework. It is a powerful, high performance Object-Relational Persistence and Query service for any Java Application. Hibernate maps Java classes to database tables and from Java data types to SQL data types and relieve the developer from 95% of common data persistence related programming tasks.

2. What is ORM?

ORM is an acronym for Object/Relational mapping. It is a programming strategy to map object with the data stored in the database. It simplifies data creation, data manipulation and data access.

3. What does an ORM solution comprises of?

a) It should have an API for performing basic CRUD (Create, Read, Update, Delete) operations on objects of persistent classes

b) Should have a language or an API for specifying queries that refer to the classes and the properties of classes

c) An ability for specifying mapping metadata

d) It should have a technique for ORM implementation to interact with transactional objects to perform dirty checking, lazy association fetching, and other optimization functions.

4. What are the different levels of ORM quality?

There are four levels defined for ORM quality.

a) Pure relational

The entire application, including the user interface, is designed around the relational model and SQL-based relational operations. It needs no extra programming or use of third party API to convert data between incompatible type systems in the object programming languages. It supports virtual object database that could be used from within programming language.

b) Light object mapping

The entities are represented as classes that are mapped manually to the relational tables. The code is hidden from the business logic using specific design patterns. This approach is successful for applications with a less number of entities, or applications with common, metadata-driven data models. This approach is most known to all.

c) Medium object mapping

The application is designed around an object model. The SQL code is generated at build time. And the associations between objects are supported by the persistence mechanism, and queries are specified using an object-oriented expression language. This is best suited for medium-sized applications with some complex transactions. Used when the mapping exceeds 25 different database products at a time.

d) Full object mapping

Full object mapping supports sophisticated object modeling: composition, inheritance, polymorphism and persistence. The persistence layer implements transparent persistence; persistent classes do not inherit any special base class or have to implement a special interface. Efficient fetching strategies and caching strategies are implemented transparently to the application.

5. What are the advantages of ORM over JDBC?

An ORM system has following advantages over plain JDBC

a) Let's business code access objects rather than DB tables.
b) Hides details of SQL queries from OO logic.
c) Based on JDBC 'under the hood'
d) No need to deal with the database implementation.
e) Entities based on business concepts rather than database structure.
f) Transaction management and automatic key generation.
g) Fast development of application.

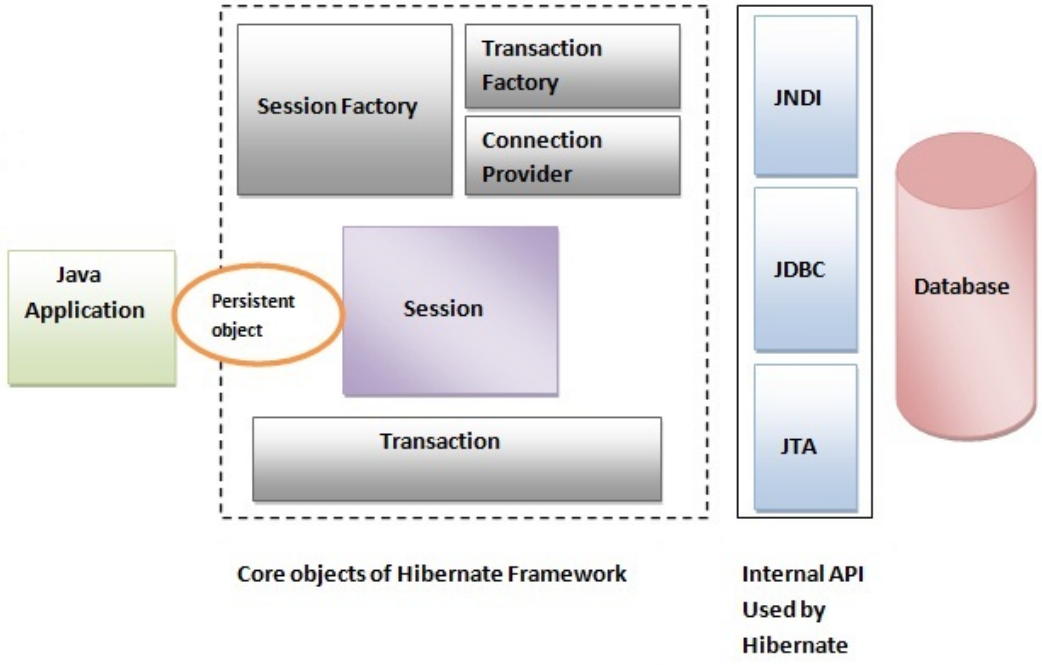
6. What are the advantages of using Hibernate?

Following are the advantages of using Hibernate.

a) Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
b) Provides simple APIs for storing and retrieving Java objects directly to and from the database.
c) If there is change in Database or in any table then the only need to change XML file properties.
d) Abstract away the unfamiliar SQL types and provide us to work around familiar Java Objects.
e) Hibernate does not require an application server to operate.
f) Manipulates complex associations of objects of your database.
g) Minimize database access with smart fetching strategies.
h) Provides simple querying of data.

7. Explain hibernate architecture?

Hibernate architecture comprises of many interfaces such as Configuration, SessionFactory, Session, Transaction etc.



Hibernate framework uses many objects session factory, session, transaction etc. alongwith existing Java API such as JDBC (Java Database Connectivity), JTA (Java Transaction API) and JNDI (Java Naming Directory Interface).

Elements of Hibernate Architecture

For creating the first hibernate application, we must know the elements of Hibernate architecture. They are as follows:

SessionFactory

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

Session

Object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

Transaction

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

ConnectionProvider

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

TransactionFactory

It is a factory of Transaction. It is optional.

4. What are the core interfaces of Hibernate?

The core interfaces of Hibernate framework are:

- Configuration
- SessionFactory
- Session
- Query
- Criteria
- Transaction

5. Is SessionFactory a thread-safe object?

Yes, SessionFactory is a thread-safe object; many threads cannot access it simultaneously.

6. What is Session?

It maintains a connection between hibernate application and database. It provides methods to store, update, delete or fetch data from the database such as persist (), update (), delete (), load (), get () etc. It is a factory of Query, Criteria and Transaction i.e. it provides factory methods to return these instances.

7. Is Session a thread-safe object?

No, Session is not a thread-safe object; many threads can access it simultaneously. In other words, you can share it between threads.

8. What is the difference between save() & saveOrUpdate()

No.	save()	saveOrUpdate()
1)	save() generates a new identifier and INSERT record into database	SaveOrUpdate can either INSERT or UPDATE based upon existence of record.
2)	Returns type is Serializable : public Serializable save(Object o)	Return type is void

9. What is the difference between session.save () and session.persist() method?

No.	save()	persist()
1)	returns the identifier (Serializable) of the instance.	return nothing because its return type is void.
2)	Syntax: public Serializable save(Object o)	Syntax: public void persist(Object o)

10. What the difference between get and load method?

The differences between get() and load() methods are given below.

No.	get()	load()
1)	Returns **null** if object is not found.	Throws **ObjectNotFoundException** if object is not found.
2)	get() method always **hit the database**.	load() method **doesn't hit** the database.
3)	It returns real object **not proxy**.	It returns **proxy object.**
4)	It should be used if **you are not sure** about the existence of instance.	It should be used if **you are sure** that instance exists.

The following Hibernate code snippet retrieves a User object from the database:

```
User user = (User) session.get(User.class, userID);
```

The get() method is special because the identifier uniquely identifies a single instance of a class. Hence it's common for applications to use the identifier as a convenient handle to a persistent object. Retrieval by identifier can use the cache when retrieving an object, avoiding a database hit if the object is already cached. The get() method returns null if the object can't be found.

Hibernate also provides a load() method:

```
User user = (User) session.load(User.class, userID);
```

If load() can't find the object in the cache or database, an exception is thrown. The load() method never returns null. The load() method may return a proxy instead of a real persistent instance. A proxy is a placeholder instance of a runtime-generated subclass (through cglib or Javassist) of a mapped persistent class, it can initialize itself if any method is called that is not the mapped database identifier getter-method.

On the other hand, get() never returns a proxy. Choosing between get() and load() is easy: If you're certain the persistent object exists, and nonexistence would be considered exceptional, load() is a good option. If you aren't certain there is a persistent instance with the given identifier, use get() and test the return value to see if it's null. Using load() has a further implication: The application may retrieve a valid reference (a proxy) to a persistent instance without hitting the database to retrieve its persistent state. So load() might not throw an exception when it doesn't find the persistent object in the cache or database; the exception would be thrown later, when the proxy is accessed.

11. How to enable hibernate SQL logging in console

Hibernate has built-in a function to enable the logging of all the generated SQL statements to the console. You can enable it by add a "show_sql" property in the Hibernate configuration file "hibernate.cfg.xml". This function is good for basic troubleshooting.

```
<property name="show_sql">true</property> //without formatting
 <property name="format_sql">true</property> //formatted sql
<property name="use_sql_comments">true</property> // with comments sql
```

12. What is the difference between update and merge method?

The differences between update() and merge() methods are given below.

No.	update() method	merge() method
1)	Update means to edit something.	Merge means to combine something.
2)	update() should be used if session doesn't contain an already persistent state with same id. It means update should be used inside the session only. After closing the session it will throw error.	merge() should be used if you don't know the state of the session, means you want to make modification at any time.

13. What the states of object are in hibernate?

There are 3 states of object (instance) in hibernate.

1. **Transient**: The object is in transient state if it is just created but has no primary key (identifier) and not associated with session.
2. **Persistent**: The object is in persistent state if session is open, and you just saved the instance in the database or retrieved the instance from the database.
3. **Detached**: The object is in detached state if session is closed. After detached state, object comes to persistent state if you call lock() or update() method.

14. What are the inheritance mapping strategies?

There are 3 ways of inheritance mapping in hibernate.

1. Table per hierarchy
2. Table per concrete class
3. Table per subclass

15. How to make immutable class in hibernate?

If you mark a class as mutable="false", class will be treated as an immutable class. By default, it is mutable="true".

16. What is automatic dirty checking in hibernate?

The automatic dirty checking feature of hibernate, calls update statement automatically on the objects that are modified in a transaction.

1. Hibernate monitors all Persistent objects
2. At the end of a unit of work, it knows which objects have been modified
3. It then calls update statements on all updated objects
4. This process of monitoring and updating only objects that have changed is called automatic dirty checking.

17. How many types of association mapping are possible in hibernate?

There can be 4 types of association mapping in hibernate.

1. One to One
2. One to Many
3. Many to One
4. Many to Many

18. Is it possible to perform collection mapping with One-to-One and Many-to-One?

No, collection mapping can only be performed with One-to-Many and Many-to-Many

19. What is lazy loading in hibernate?

Loading the object the Lazy loading. For example an individual's on demand is called profile in an online store may have multiple delivery addresses, Which means Addresses are associated to User a **One-to-Many** relationship, so loading the Address of an user when it is required is called the lazy loading. Since Hibernate 3, lazy loading is enabled by default you don't need to do lazy="true".

Also note that @OneToMany and @ManyToMany associations are defaulted to **LAZY loading**; and @OneToOne and @ManyToOne are defaulted to **EAGER loading**. This is important to remember to avoid any pitfall in future.

Lazy loading in hibernate improves the performance. It loads the child objects on demand.

20. What is HQL (Hibernate Query Language)?

Hibernate Query Language is known as an object oriented query language. It is like structured query language (SQL).The main advantage of HQL over SQL is:

1. You don't need to learn SQL
2. Database independent
3. Simple to write query

21. What is the difference between first level cache and second level cache?

No.	First Level Cache	Second Level Cache
1)	First Level Cache is **associated with Session**.	Second Level Cache is associated with **SessionFactory**.
2)	It is **enabled** by default.	It is **not enabled** by default.

22. What the two key components are of hibernate configuration object?

The Configuration object provides two keys components:

a) **Database Connection:** This is handled through one or more configuration files supported by Hibernate. These files are hibernate.properties and hibernate.cfg.xml.

b) Class Mapping Setup: This component creates the connection between the Java classes and database tables.

23. What is the purpose of Session.beginTransaction() method?

Session.beginTransaction method begins a unit of work and returns the associated Transaction object.

24. Which method is used to add a criteria to a query?

Session.createCriteria creates a new Criteria instance, for the given entity class, or a superclass of an entity class.

25. Which method is used to create a HQL query?

Session.createQuery creates a new instance of Query for the given HQL query string

26. Which method is used to create a SQL query?

Session.createSQLQuery creates a new instance of SQLQuery for the given SQL query string.

27. Which method is used to remove a persistent instance from the datastore?

Session.delete removes a persistent instance from the datastore.

28. Which method is used to get a persistent instance from the datastore?

Session.get returns the persistent instance of the given named entity with the given identifier, or null if there is no such persistent instance.

29. **What is named SQL query in Hibernate?**

Named queries are SQL queries which are defined in mapping document using <sql-query> tag and called using Session.getNamedQuery() method. Named query allows you to refer a particular query by the name you provided, by the way, you can define named query in hibernate either by using annotations or XML mapping file, as I said above. @NameQuery is used to define single named query and @NameQueries is used to define multiple named query in hibernate

30. **What is SessionFactory in Hibernate?** Is **SessionFactory thread-safe?**

SessionFactory, as the name suggest, is a factory to hibernate Session objects. SessionFactory is often built during start-up and used by application code to get session object. It acts as a single data store and it's also thread-safe so that multiple threads can use the same SessionFactory. Usually, a Java JEE application has just one SessionFactory, and individual threads, which are servicing client's request obtain hibernate Session instances from this factory, that's why any implementation of SessionFactory interface must be thread-safe. Also, the internal state of SessionFactory, which contains all metadata about Object/Relational mapping is Immutable and cannot be changed once created.

31. **What is Session in Hibernate? Can we share single Session among multiple threads in Hibernate?**

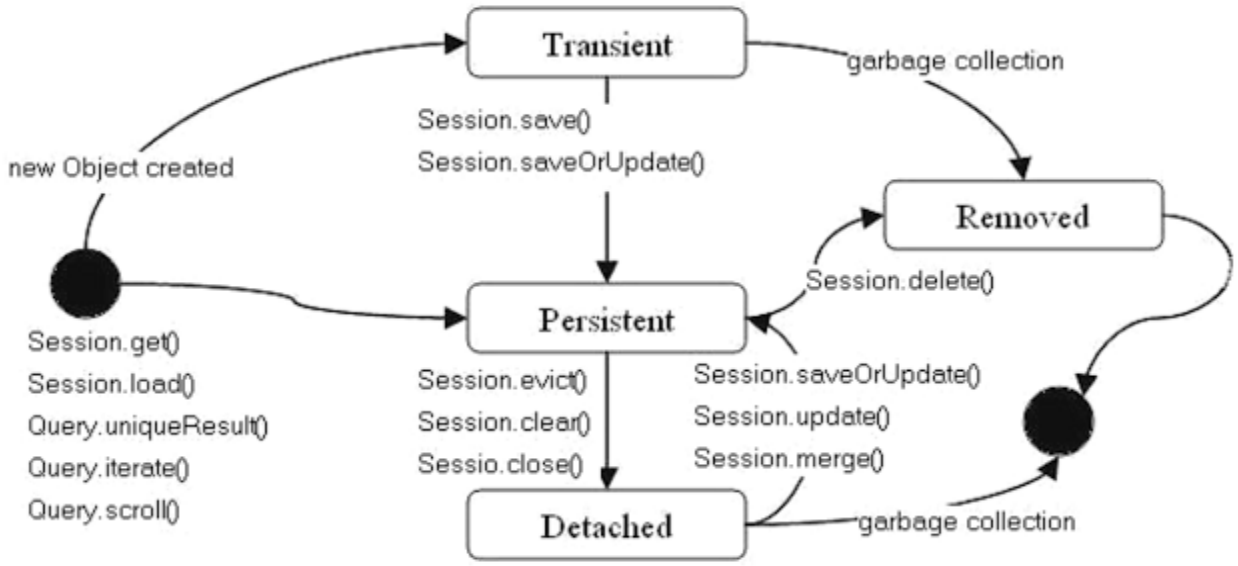
After SessionFactory its time for Session. Session represent a small unit of work in Hibernate, they maintain a connection with the database and they are not thread-safe, it means you cannot share Hibernate Session between multiple threads. Though Session obtains database connection lazily it's good to close session as soon as you are done with it.

32. **What is the difference between sorted and ordered collection in hibernate?**

A sorted collection is sorted in memory by using Java Comparator while an ordered collection uses database's order by clause for ordering. For large data set it's better to use ordered collection to avoid any OutOfMemoryError in Java, by trying to sort them in memory.

33. **What is the difference between transient, a persistent and detached object in Hibernate?**

In Hibernate, Object can remain in three state transient, persistent or detached. An object which is associated with Hibernate session is called persistent object. Any change in this object will reflect in the database based on your flush strategy i.e. automatic flush whenever any property of object change or explicit flushing by calling Session.flush() method. On the other hand, if an object which is earlier associated with Session, but currently not associated with it are called detached object.



34. **What does Session lock() method do in Hibernate OR what is difference between Session's lock() and update() method?**

Session's lock() method reattach object without synchronizing or updating with the database. So you need to be very careful while using lock() method. By the way, you can always use Session's update() method to sync with the database during reattachment.

35. **What is Second-level Cache in Hibernate?**

Second-level Cache is maintained at SessionFactory level and can improve performance by saving few database round trip. Another worth noting point is that second level cache is available to the whole application rather than any particular session.

36. **Why it's important to provide no argument constructor in Hibernate Entities?**

Every Hibernate Entity class must contain a no argument constructor, because Hibernate framework creates an instance of them using Reflection API, by calling Class.newInstance() method. This method will throw InstantiationException if it doesn't found any argument constructor inside Entity class.

37. What is query cache in Hibernate?

QueryCache actually stores the result of SQL query for future calls. Query cache can be used along with second level cache for improved performance. Hibernate support various open source caching solution to implement Query cache e.g. EhCache.

38. Can we make a Hibernate Entity Class final?

Yes, you can make a Hibernate Entity class final, but that's not a good practice. Since Hibernate uses a proxy pattern for performance improvement in the case of the lazy association, by making an entity final, Hibernate will no longer be able to use a proxy, because Java doesn't allow extension of the final class, thus limiting your performance improvement options. Though, you can avoid this penalty if your persistent class is an implementation of an interface, which declares all public methods defined in the Entity class.

39. Java Annotation (is the part of JAVA 5) – replaces the mapping XML file

Some of the example Annotation are below:

```
@Entity
@Table(name = "EMPLOYEE")

@Id @GeneratedValue
@Column(name = "id")
```

Package: javax.persistence

40. What jars are to be needed for Hibernate Annotation?

a) hibernate-commons-annotations.jar
b) ejb3-persistence.jar
c) hibernate-annotations.jar

41. What are the core annotation used in hibernate?

a) @Entity is used to mark the class as an Entity Bean. Class should have at least a package scope no-argument constructor.
b) Table is used to specify the table to persist the data. name attribute refers table name.
c) If @Table is not defined then Hibernate will by default use class name as table name.
d) @Id is the identifier property of Entity Bean. It determine the default access strategy that hibernate will use for mapping. If it is placed over field - field access will use.
e) If it is placed over setter method – property access will use.
f) @GeneratedValue is used to specify primary key generation strategy. By default Auto is used.
g) @Column is used to specify a column to which a field / property will mapped. If not defined by default property name will be used as column name.

42. When to use Annotation?

a. If we have flexibility to use Java 5 Environment
b. If we know which production database will use. In order to support multiple database it's better to have mapping information into separate xml files rather than using Annotations. We can have multiple xml files each specific to particular database.

43. What is HQL?

Hibernate Query Language (HQL) is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries.

44. What are the Named Paramters?

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. Following is the simple syntax of using named parameters:

For Ex:

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

45. Pagination using Query

There are two methods of the Query interface for pagination.

a) **Query setFirstResult(int startPosition)**
   This method takes an integer that represents the first row in your result set, starting with row 0.

b) **Query setMaxResults(int maxResult)**
   This method tells Hibernate to retrieve a fixed number **maxResults** of objects.

46. Criteria

What is Criteria?

Criteria is used to retrieve/update the data based on some criteria.

1. We can add Restriction to the criteria using add() method
   ```
   Criteria cr = session.createCriteria(Employee.class);
   cr.add(Restrictions.eq("salary", 2000));
   List results = cr.list();
   ```
2. You can create AND or OR conditions using LogicalExpression restrictions as follows:
3. Pagination using Criteria
4. Sorting the Results
   ```
   // To sort records in descening order
   crit.addOrder(Order.desc("salary"));
   ```

47. Caching

What is Caching?

Caching is all about application performance optimization and it sits between your application and the database to avoid the number of database hits as many as possible to give a better performance for performance critical applications.

a) First-level cache:

The first-level cache is the Session cache and is a mandatory cache through which all requests must pass.
The Session object keeps an object under its own power before committing it to the database.
If you issue multiple updates to an object, Hibernate tries to delay doing the update as long as possible to reduce the number of update SQL statements issued. If you close the session, all the objects being cached are lost and either persisted or updated in the database.

b) Second-level cache:

Second level cache is an optional cache and first-level cache will always be consulted before any attempt is made to locate an object in the second-level cache. The second-level cache can be

configured on a per-class and per-collection basis and mainly responsible for caching objects across sessions.
The Second Level Cache: Can be enabled by placing property in hbm (hibernate mapping file) as below, to use second level we need to decide based on some concurrency strategy

<cache usage="read-write"/>

Any third-party cache can be used with Hibernate. An org.hibernate.cache.CacheProvider interface is provided, which must be implemented to provide Hibernate with a handle to the cache implementation.

c) Query-level cache:

Hibernate also implements a cache for query result sets that integrates closely with the second-level cache. To use query level cache we need to set the following tag in the **hibernate.cfg.xml file**

use_query_cache="true"

Or in java code as

```
Session session = SessionFactory.openSession();
Query query = session.createQuery("FROM EMPLOYEE");
query.setCacheable(true);
query.setCacheRegion("employee");
List users = query.list();
SessionFactory.closeSession();
```

48. Batch Processing

Consider a situation when you need to upload a large number of records into your database using Hibernate. Following is the code snippet to achieve this using Hibernate:

Because by default, Hibernate will cache all the persisted objects in the session-level cache and ultimately your application would fall over with an OutOfMemoryException somewhere around the 50,000th row. You can resolve this problem if you are using batch processing with Hibernate.

To enable batch processing in configuration xml file as below: in **hibernate.cfg.xml**

```
<property name="hibernate.jdbc.batch_size">
    50
</property>
```

To use the batch processing feature, first set **hibernate.jdbc.batch_size** as batch size to a number either at 20 or 50 depending on object size. This will tell the hibernate container that every X rows to be inserted as batch. To implement this in your code we would need to do little modification as follows:

```
Session session = SessionFactory.openSession();
Transaction tx = session.beginTransaction();
for ( int i=0; i<100000; i++ ) {
   Employee employee = new Employee(.....);
   session.save(employee);
       if( i % 50 == 0 ) { // Same as the JDBC batch size
      //flush a batch of inserts and release memory:
      session.flush();
      session.clear();
   }
```

```
}
tx.commit();
session.close();
```

flush() and **clear()** methods available with Session object so that Hibernate keep writing these records into the database instead of caching them in the memory.

49. Hibernate call store procedure

In Hibernate, there are three approaches to call a database store procedure.

1. Native SQL – createSQLQuery

```
Query query = session.createSQLQuery(
        "CALL GetStocks(:stockCode)")
        .addEntity(Stock.class)
        .setParameter("stockCode", "7277");

List result = query.list();
for(int i=0; i<result.size(); i++){
        Stock stock = (Stock)result.get(i);
        System.out.println(stock.getStockCode());
}
```

2. NamedNativeQuery in annotation

```
//Stock.java
...
@NamedNativeQueries({
        @NamedNativeQuery(
        name = "callStockStoreProcedure",
        query = "CALL GetStocks(:stockCode)",
        resultClass = Stock.class
        )
})

@Entity
@Table(name = "stock")
public class Stock implements java.io.Serializable {
```

Call it with **getNamedQuery()**.

```
Query query = session.getNamedQuery("callStockStoreProcedure")        .setParameter("stockCode",
"7277");
List result = query.list();
for(int i=0; i<result.size(); i++){
        Stock stock = (Stock)result.get(i);
        System.out.println(stock.getStockCode());
}
```

3. sql-query in XML mapping file

Declare your store procedure inside the "sql-query" tag.

```
<hibernate-mapping>
    <class name="com.mkyong.common.Stock" table="stock" ...>
        <id name="stockId" type="java.lang.Integer">
            <column name="STOCK_ID" />
            <generator class="identity" />
        </id>
        <property name="stockCode" type="string">
```

```
            <column name="STOCK_CODE" length="10" not-null="true" unique="true" />
        </property>
        ...
    </class>

    <sql-query name="callStockStoreProcedure">
         <return alias="stock" class="com.mkyong.common.Stock"/>
         <![CDATA[CALL GetStocks(:stockCode)]]>
    </sql-query>

</hibernate-mapping>
```

Call it with getNamedQuery().

```
Query query = session.getNamedQuery("callStockStoreProcedure")        .setParameter("stockCode",
"7277");
List result = query.list();
for(int i=0; i<result.size(); i++){
        Stock stock = (Stock)result.get(i);
        System.out.println(stock.getStockCode());
}
```