3 Approaches Calling stored procedure in Hibernate MySQL store procedure, which accept a stock code parameter and return stock data.

**DELIMITER $$**

**CREATE PROCEDURE `GetStocks`(int\_stockcode varchar(20))**

**BEGIN**

**SELECT \* FROM stock where stock\_code = int\_stockcode;**

**END $$**

**DELIMITER ;**

In MySQL, you can simple call it with a **call** keyword: CALL GetStocks('7277');

1. Native SQL – createSQLQuery

You can use **createSQLQuery()** to call a store procedure directly.

Query query = session.createSQLQuery("CALL GetStocks(:stockCode)")

.addEntity(Stock.class) .setParameter("stockCode", "7277");

List result = query.list();

2. NamedNativeQuery in annotation

Declare your store procedure inside the **@NamedNativeQueries** annotation.

@NamedNativeQueries({ //Stock.java

@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();

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();

There are not much big different between the three approaches, which method you choose is depend on your personal prefer.

From the **JPA 2.0 spe**c, the defaults are like so:

OneToMany: LAZY

ManyToOne: EAGER

ManyToMany: LAZY

OneToOne: EAGER

And in **hibernate, all is Lazy**

From Hibernate Docs,

By default, Hibernate uses lazy select fetching for collections and lazy proxy fetching for single-valued associations. These defaults make sense for most associations in the majority of applications.

To answer your question, Hibernate is an implementation of the JPA standard. Hibernate has its own quirks of operation, but as per the Hibernate docs

So Hibernate will always load any object using a lazy fetching strategy, no matter what type of relationship you have declared.

**JPA Spec** assumes that in general most of the applications will require the singleton relations by default be eager, whereas multi value relations by default be lazy.

Refer [here](https://docs.jboss.org/hibernate/orm/3.6/reference/en-US/html/performance.html#performance-fetching-lazy) (referring chapter 21 already in a doc) for more