**Hibernate named query examples**

Often times, developer like to put HQL string literals scatter all over the Java code, this method is hard to maintaine and look ugly. Fortunately, Hibernate come out a technique called “**names queries**” , it lets developer to put all HQL into the XML mapping file or via annotation.

**How to declare named query**

The named query is supported in both HQL or native SQL. see examples…

**1. XML mapping file**

**HQL in mapping file**

<!-- stock.hbm.xml -->

<hibernate-mapping>

<class name=*"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>

<query name=*"findStockByStockCode"*>

<![CDATA[from Stock s where s.stockCode = :stockCode]]>

</query>

</hibernate-mapping>

**Native SQL in mapping file**

<!-- stock.hbm.xml -->

<hibernate-mapping>

<class name=*"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=*"findStockByStockCodeNativeSQL"*>

<return alias=*"stock"* class=*" Stock"* />

<![CDATA[select \* from stock s where s.stock\_code = :stockCode]]>

</sql-query>

</hibernate-mapping>

You can place a named query inside ‘**hibernate-mapping**‘ element, but do not put before the ‘**class**‘ element, Hibernate will prompt invalid mapping file, all your named queries have to put after the ‘**class**‘ element.

**Note**  
Regarding the CDATA , it’s always good practice to wrap your query text with CDATA, so that the XML parser will not prompt error for some special XML characters like ‘>’ , <‘ and etc.

## 2. Annotation

**HQL in annotation**

@NamedQueries({

@NamedQuery(

name = "findStockByStockCode",

query = "from Stock s where s.stockCode = :stockCode"

)

})

@Entity

@Table(name = "stock", catalog = "mkyong")

**public** **class** Stock **implements** java.io.Serializable {

**Native SQL in annotation**

@NamedNativeQueries({

@NamedNativeQuery(

name = "findStockByStockCodeNativeSQL",

query = "select \* from stock s where s.stock\_code = :stockCode",

resultClass = Stock.**class**

)

})

@Entity

@Table(name = "stock", catalog = "mkyong")

**public** **class** Stock **implements** java.io.Serializable {

In native SQL, you have to declare the ‘**resultClass**‘ to let Hibernate know what is the return type, failed to do it will caused the exception “**org.hibernate.cfg.NotYetImplementedException: Pure native scalar queries are not yet supported**“.

**Call a named query**

In Hibernate, you can call the named query via **getNamedQuery** method.

Query query = session.getNamedQuery("findStockByStockCode")

.setString("stockCode", "7277");

Query query = session.getNamedQuery("findStockByStockCodeNativeSQL")

.setString("stockCode", "7277");

**Conclusion**

Named queries are global access, which means the name of a query have to be unique in XML mapping files or annotations. In real environment, it’s always good practice to isolate all the named queries into their own file. In addition, named queries stored in the Hibernate mapping files or annotation are more easier to maintain than queries scattered through the Java code.