Hibernate Filters

**What is a Hibernate Filter?**

Hibernate3 provides an innovative new approach to handling data with "visibility" rules. A Hibernate filter is a global, named, parameterized filter that can be enabled or disabled for a particular Hibernate session.

Hibernate3 has the ability to pre-define filter criteria and attach those filters at both a class level and a collection level. A filter criteria allows you to define a restriction clause similar to the existing "where" attribute available on the class and various collection elements.

**Why, Filter?**

With Hibernate3 there is a new way to filtering the results of searches. Sometimes it is required to only process a subset of the data in the underlying Database tables. Hibernate filters are very useful in those situations. Other approaches for these kind of problems is to use a database view or use a WHERE clause in the query or Hibernate Criteria API.

This can also be done using a WHERE clause within the SQL SELECT query or Hibernate's HQL SELECT query. For a small application, it is okay to do this, but for a large and complex application it might be a troublesome effort. Moreover, it will be like searching each and every SQL query and making the changes in the existing code which have been thoroughly tested.

One of the other use cases of filters may be as the user's view of the organization data. A user can only view the data that he/she is authorized to.

**Hibernate Filter?**

How to use hibernate filter in application having hibernate support:

1. **Required jars**
2. **Filters component**
3. **XML based configuration**
4. **Annotations based configuration**
5. **Enable/Disable filter**
6. **Required Jars**

It needs to download hibernate3.jar under Hibernate Core, which contains filter support for hibernate based applications

1. **Filter Components**

Filters comprises of two components Filter definition & filter declaration for entity where is has to apply. Filter can be applied to entity or collections.

1. **XML based configuration**

In order to use filters, they must first be defined and then attached to the appropriate mapping elements. To define a filter, use the <filter-def/> element within a <hibernate-mapping/> element.

These filter definitions must contain the name of the filter and the names and types of any filter parameters. Specify filter parameters with the <filter-param> XML element. Filter parameters are similar to the named parameters for HQL queries. We need to specify a ':' (colon) before the parameter name/condition. Here is the mapping file from the sample code.

<filter-def name="myFilter" condition = "param > :myFilterParam">

<filter-param name="myFilterParam" type="string"/>

</filter-def>

The above condition defined will be applied to all filters.

Now attach the filters to class or collection mapping elements. You can attach a single filter to more than one class or collection. To do this, you add a <filter> XML element to each class or collection. The <filter> XML element has two attributes viz. name and condition. The name references a filter definition (in the sample application it's : statusFilter) while condition is analogous to a WHERE clause in HQL.

This filter can then be attached to a class:

<class name="myClass" ...>

<filter name="myFilter" condition=":myFilterParam = MY\_FILTERED\_COLUMN"/>

</class>

Or, to a collection:

<set ...>

<filter name="myFilter" condition=":myFilterParam = MY\_FILTERED\_COLUMN"/>

</set>

Or, to both or multiples of each at the same time.

Filters once defined can be used multiple times. Filters-def is not required to defined every time a class is using. This filter-def can be applied to all the classes by just using <filter name=” … ”>.

1. **Annotaions Based configuration**

@FilterDef annotation is used to define a filter as <filer-def> in xml. It defines a name, default condition and parameter types (if any).

@FilterDef(name = " myFilter ",

defaultCondition = "col-name = : myFilterParam ",

parameters = {@ParamDef(name = "myFilterParam", type = "integer")})

Like-wise we can define multiple parameters for filter definition by passing an array of @ParamDef.

The @Filter annotation is another way to filter out entities or collections using a custom SQL criteria, for both entities and collections. Unlike the @Where annotation, @Filter allows you to parameterize the filter clause at runtime. We can use any number of filter defined in an entity. A @Filter must associate to any @FilterDef name;

@Filter(name = " myFilter ", condition = "col-name = : myFilterParam " )

1. **Enable/Disable Filter**

In the java code, we can programmatically enable or disable the filter. By default the Hibernate Session doesn't have any filters enabled on it.

The Session interface contains the following methods:

* public Filter enableFilter(String filterName)
* public Filter getEnabledFilter(String filterName)
* public void disableFilter(String filterName)

The Filter interface contains some of the important methods:

* public Filter setParameter(String name, Object value)
* public Filter setParameterList(String name, Collection values)
* public Filter setParameterList(String name, Object[] values)

setParameter() method is mostly used. Be careful and specify only the type of java object that you have mentioned in the parameter at the time of defining filter in the mapping file.

The two setParameterList() methods are useful for using IN clauses in your filters. If you want to use BETWEEN clauses, use two different filter parameters with different names.

Session session = sessionFactory.openSession();

//enabling filter myFilter

Filter filter = session.enableFilter("myFilter");

filter.setParameter("myFilterParam ", param-value);

***For References:***

<http://www.javaworld.com/article/2077880/data-storage/data-storage-introduction-to-hibernate-search.html>

<https://docs.jboss.org/hibernate/search/4.2/reference/en-US/html/search-configuration.html#infinispan-directories>

<https://dzone.com/refcardz/getting-started-with-hibernate>