**INTERNITY FOUNDATION**

**TASK-12**

**Submitted By:**

**Amisha Singhal**

**Java Batch**

**Hibernate Configuration Hibernate Sessions Annotation Based Configuration**

**Ans-**Hibernate can operate in different environment, it requires a wide range of configuration parameters. These configurations contain the mapping information that provides different functionalities to Java classes. Generally, we provide database related mappings in the configuration file. Hibernate facilitates to provide the configurations either in an XML file (like hibernate.cfg.xml) or properties file (like hibernate.properties).

An instance of Configuration class allows specifying properties and mappings to applications. This class also builds an immutable SessionFactory.

We can acquire the Configuration class instance by instantiating it directly and specifying mappings in the configuration file. Use the addResource() method, if the mapping files are present in the classpath.

**Configuration cfg = new Configuration()**

**.addResource("employee.hbm.xml")**

**Hibernate O/R Mappings**

**Ans-**

<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name="roseindia.tutorial.hibernate.Contact" table="CONTACT">

<id name="id" type="long" column="ID" >

<generator class="assigned"/>

</id>

<property name="firstName">

<column name="FIRSTNAME" />

</property>

<property name="lastName">

<column name="LASTNAME"/>

</property>

<property name="email">

<column name="EMAIL"/>

</property>

</class>

</hibernate-mapping>

Hibernate mapping documents are simple xml documents. Here are important elements of the mapping file:.

1. **<hibernate-mapping> element**  
   The first or root element of hibernate mapping document is <hibernate-mapping> element. Between the <**hibernate-mapping>** tag class element(s) are present.
2. **<class> element**  
   The <Class> element maps the class object with corresponding entity in the database. It also tells what table in the database has to access and what column in that table it should use. Within one <hibernate-mapping> element, several <class> mappings are possible.
3. **<id> element**  
   The <id> element in unique identifier to identify and object. In fact <id> element map with the primary key of the table. In our code :  
   <id name="id" type="long" column="ID" >  
   primary key maps to the **ID** field of the table **CONTACT**. The attributes of the id element are:
   * **name:**The property name used by the persistent class.
   * **column:** The column used to store the primary key value.
   * **type:** The Java data type used.
   * **unsaved-value:** This is the value used to determine if a class has been made persistent. If the value of the id attribute is null, then it means that this object has not been persisted.
4. **<generator> element**  
   The **<generator>**methodis used to generate the primary key for the new record. Here is some of the commonly used generators :  
      
   **\*** **Increment**- This is used to generate primary keys of type long, short or int that are unique only. It should not be used in the clustered deployment environment.  
      
   **\*  Sequence** - Hibernate can also use the sequences to generate the primary key. It can be used with DB2, PostgreSQL, Oracle, SAP DB databases.  
       
   **\* Assigned** - Assigned method is used when application code generates the primary key.
5. **<property> element**  
   The property elements define standard Java attributes and their mapping into database schema. The property element supports the column child element to specify additional properties, such as the index name on a column or a specific column type.

**Inheritence Mappings**

**Ans-**We can map the inheritance hierarchy classes with the table of the database. There are three inheritance mapping strategies defined in the hibernate:

1. **Table Per Hierarchy**
2. **Table Per Concrete class**
3. **Table Per Subclass**

#### Table Per Hierarchy

In table per hierarchy mapping, single table is required to map the whole hierarchy, an extra column (known as discriminator column) is added to identify the class. But null-able values are stored in the table.

#### Table Per Concrete class

In case of table per concrete class, tables are created as per class. But duplicate column is added in subclass tables.

#### Table Per Subclass

In this strategy, tables are created as per class but related by foreign key. So there are no duplicate columns.

**Hibernate Query Language**

**Ans-** Hibernate Query Language (HQL) is same as SQL (Structured Query Language) but it doesn't depends on the table of the database. Instead of table name, we use class name in HQL. So it is database independent query language.

HQL works with persistent objects and their properties. HQL queries are translated by Hibernate into conventional SQL queries, which in turns perform action on database.

Although we can use SQL statements directly with Hibernate using Native SQL, we can recommend to use HQL whenever possible to avoid database portability hassles. Keywords like SELECT, FROM, and WHERE, etc., are not case sensitive, but properties like table and column names are case sensitive in HQL.

**Advantage of HQL**

* database independent
* supports polymorphic queries
* easy to learn for Java Programmer

**Example of HQL to get all the records**

**Query query=session.createQuery("from Emp");**

**//here persistent class name is Emp**

**List list=query.list();**

**Criteria Queries**

**Ans-**Criteria API allows us to build up a criteria query object programmatically where we can apply filtration rules and logical conditions.

The Hibernate **Session** interface provides **createCriteria()** method, which can be used to create a **Criteria** object that returns instances of the persistence object's class when your application executes a criteria query.

## The Criteria API provides the org.hibernate.criterion.Order class to sort your result set in either ascending or descending order, according to one of your object'sproperties.

## Creating a Criteria instance

The interface org.hibernate.Criteria represents a query against a particular persistent class. The Session is a factory for Criteria instances.

**Criteria crit = sess.createCriteria(Cat.class);**

**crit.setMaxResults(50);**

**List cats = crit.list();**

**JPA**

**Ans-** The Java Persistence API (JPA) is a specification of Java. It is used to persist data between Java object and relational database. JPA acts as a bridge between object-oriented domain models and relational database systems.

It requires an implementation. So, ORM tools like Hibernate, TopLink and iBatis implements JPA specifications for data persistence.

To reduce the burden of writing codes for relational object management, a programmer follows the ‘JPA Provider’ framework, which allows easy interaction with database instance. Here the required framework is taken over by JPA.

JPA permits the developer to work directly with objects rather than with SQL statements. The JPA implementation is typically called persistence provider.

The mapping between Java objects and database tables is defined via persistence metadata. The JPA provider will use the persistence metadata information to perform the correct database operations.JPA metadata is typically defined via annotations in the Java class. Alternatively, the metadata can be defined via XML or a combination of both. A XML configuration overwrites the annotations.JPA defines a SQL-like Query language for static and dynamic queries.