Hibernate

* Hibernate is a high-performance Object/Relational persistence
* Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities.
* Hibernate maps Java classes to database tables and from Java data types to SQL data types.

**Hibernate Advantages:**

* Hibernate takes care of mapping Java classes to database tables using XML files and without writing any line of code.
* Provides simple APIs for storing and retrieving Java objects directly to and from the database.
* If there is change in Database or in any table then the only need to change XML file properties.
* Minimize database access with smart fetching strategies.
* Provides Simple querying of data.

**What is JDBC?**

* JDBC stands for Java Database Connectivity and provides a set of Java API for accessing the relational databases from Java program.
* These Java APIs enables Java programs to execute SQL statements and interact with any SQL compliant database.
* JDBC provides a flexible architecture to write a database independent application that can run on different platforms and interact with different DBMS without any modification.

**Disadvantage:**

* Complex if it is used in large projects
* Large programming overhead
* No encapsulation
* Hard to implement MVC concept
* Query is DBMS specific

**Why Object Relational Mapping (ORM)?**

* **Inheritance:** RDBMSs do not define anything similar to Inheritance which is a natural paradigm in object-oriented programming languages.
* I**dentity:** A RDBMS the primary key alone for uniqueness but Java has both object identity (a==b) and object equality (a.equals(b)).
* **Associations:** Object-oriented languages uses object references where as am RDBMS uses a foreign key column for association.

**What is ORM?**

ORM stands for Object-Relational Mapping (ORM) is a programming technique for converting data between relational databases and object oriented programming languages such as Java

**Hibernate - Architecture**

The Hibernate architecture is layered to keep you isolated from having to know the underlying APIs.

Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.

**Hibernate has five objects:**

* Configuration
* Session Factory
* Session
* Transaction
* Criteria and Query

**Configuration Object:**

The Configuration object is the first Hibernate object you create in any Hibernate application and usually created only once during application initialization.

It provides two keys components:

* **Database Connection:** like hibernate.properties and hibernateSessionFactory.xml.
* **Class Mapping Setup:** like Users.hbm.xml

**SessionFactory Object:**

* Configuration object is used to create a SessionFactory object which inturn configures Hibernate using the supplied configuration file and allows for a Session object to be instantiated.
* The SessionFactory is a thread safe object and used by all the threads of an application.
* The SessionFactory is heavyweight object so usually it is created during application start up and kept for later use.
* if you are using multiple databases then you would have to create multiple SessionFactory objects.

**Session Object:**

* A Session is used to get a physical connection with a database.
* The Session object is lightweight and designed to be instantiated each time when needed
* The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed as needed.

**Transaction Object:**

* A Transaction represents a unit of work with the database and are handled by an underlying transaction manager.
* In such case, if one step fails, the whole transaction fails (which is termed as atomicity)
* This is an optional object.
* A transaction is associated with Session and instantiated by calling session.beginTransaction().

**Query Object:**

* Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects.
* A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

**Criteria Object:**

* Criteria object are used to create and execute object oriented criteria queries to retrieve objects based on the specific criteria.

Eg: Retrieving all the records of a table whose salary is greater than 50000.

**Hibernate Properties:**

Following is the list of important properties you would require to configure for a database in a standalone situation:

1. hibernate.dialect - This property makes Hibernate generate the appropriate SQL for the chosen database.
2. hibernate.connection.driver\_class - the JDBC driver class.
3. hibernate.connection.url - the JDBC URL to the database instance.
4. hibernate.connection.username - the database username.
5. hibernate.connection.password - the database password.
6. hibernate.connection.pool\_size - limits the number of connections waiting in the Hibernate database connection pool.
7. hibernate.connection.autocommit - allows auto-commit mode to be used for the JDBC connection.

**Instances of session object:**

**transient:** A new instance of a persistent class which is not associated with a Session and has no representation in the database and no identifier value is considered transient by Hibernate.

**persistent:** You can make a transient instance persistent by associating it with a Session. Java classes whose objects or instances will be stored in database tables are called persistent classes in Hibernate.

**detached:** Once we close the Hibernate Session, the persistent instance will become a detached instance.

**Hibernate Mapping POJO:**

A mapping document helps Hibernate in determining how to pull the values from the classes and map them with table and associated fields.

Hibernate works best if these classes follow some simple rules, also known as the Plain Old Java Object (POJO) programming model.

There are following main rules of persistent classes, however, none of these rules are hard requirements.

* All Java classes that will be persisted need a default constructor.
* All classes should contain an ID and this property maps to the primary key column of a database table.
* All attributes should be declared private and have getXXX and setXXX methods defined in the JavaBean style.

**Hibernate Mapping file:** are usually defined in an XML document and this mapping file instructs Hibernate how to map the defined class to the database tables.

Tools to generate the mapping document include XDoclet, Middlegen and AndroMDA for advanced Hibernate users.

You should save the mapping document in a file with the format <classname>.hbm.xml.

Eg: <hibernate-mapping>

<class name="Employee" table="EMPLOYEE">

<meta attribute="class-description">

This class contains the employee detail.

</meta>

<id name="id" type="int" column="id">

<generator class="native"/>

</id>

<property name="firstName" column="first\_name" type="string"/>

<property name="lastName" column="last\_name" type="string"/>

<property name="salary" column="salary" type="int"/>

</class>

</hibernate-mapping>

**Hibernate generator:**

* uses assigned (default), sequence, increment or hilo algorithm based on dB
* hibernate provides different primary key generator classes and all these classes are implemented from **org.hibernate.id.IdentifierGeneratar** Interface

**O/R mapping using hibernate:**

There are three types of mapping in hibernate.

**Collections Mappings:**

If an entity or class has collection of values for a particular variable, then we can map those values using any one of the collection interfaces available in java.

We can use user-defined collection interfaces to map as well but is not recommended.

**List:** Ordered collection of entities, duplicate allowed. The index column will need for mapping.

**Set:** Unordered collection of unique entities, duplicates not allowed. It is important to override GetHashCode and Equals to indicate the business definition of duplicate.

**Bag:** Unordered list of entities, duplicates allowed. The index column of the list is not needed.

Eg: private List<String> answers;//List can be of any type

can be represented as

<list name="answers" table="ans100">

<key column="qid"></key>

<index column="type"></index>

<element column="answer" type="string"></element>

</list>

**Association Mappings:**

An association mapping can be unidirectional as well as bidirectional.

* Many-to-One
* One-to-One
* One-to-Many
* Many-to-Many

**Component Mappings:**

An component is an object that is stored as an value rather than entity reference.

A Component mapping is a mapping for a class having a reference to another class as a member variable (ie) in case of HAS\_A relation.

This is mainly used if the dependent object doesn’t have primary key.

Tag used: <component>

**Scenario:** For an employee class the address is a class which contains street name, city, etc., which depends only on the employee class.

**Hibernate - Annotations**

**Requirements:** JDK 5.0 and hibernate 3.X

* **@Entity** - marks the class as an entity bean.
* **@Table** - allows to specify the details of the table.
* **@Id and @GeneratedValue** – represents the primary key. @Id annotation will automatically determine the most appropriate primary key generation strategy. but can override this by applying the @GeneratedValue
* **@Column** - specify the details of the column. Attributes are name, length, nullable and unqiue

**Hibernate Query Language (HQL):**

is an object-oriented query language, similar to SQL, but instead of operating on tables and columns, HQL works java objects and their properties

is useful because it is database independent. (ie) no need in changing query if we change the dB.

**Pagination using Query**

There are two methods of the Query interface for pagination.

setFirstResult(int startPosition)

setMaxResults(int maxResult)

eg: Query query = session.createQuery(hql);

query.setFirstResult(1);

query.setMaxResults(10);

**Hibernate - Caching**

Caching sits between your application and the database to avoid the number of database hits as many as possible to give a better performance.

**First-level cache:**

* The first-level cache is associated with the Session object and is enabled by default.
* 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.

**Second-level cache:**

* Second level cache is an optional cache and is associated with session factory object
* First-level cache will always be consulted before an object is located in the second-level cache.
* The second-level cache can be configured on a per-class and per-collection basis.

The Hibernate second-level cache is set up in two steps.

**1. Concurrency strategies:**

A concurrency strategy is a mediator which responsible for storing and retrieving items from the cache.

Some strategies are:

* Transactional
* Read-write
* Nonstrict-read-write
* Read-only

eg: <cache usage="read-write"/>

is specified in <hibernate-mapping>

**2. Cache provider:**

Used to configure cache expiration and physical cache attributes.

Eg: <property name="hibernate.cache.provider\_class">

org.hibernate.cache.EhCacheProvider

</property>

is specified in hibernate config file.

**Query-level cache:**

* Hibernate also implements a cache for query resultsets that integrates closely with the second-level cache.
* This is an optional feature and is only useful for queries that are run frequently with the same parameters.

**The Hibernate Query-level cache is set up in two steps:**

* you must first activate it using the hibernate.cache.use\_query\_cache="true" property in the configuration file
* use the query cache, you use the setCacheable(Boolean) method of the Query class

eg: Query query = session.createQuery("FROM EMPLOYEE");

query.setCacheable(true);

**Hibernate - Batch Processing**

Hibernate will cache all the persisted objects in the session-level cache and ultimately your application would fall over with an OutOfMemoryException while inserting a large number of record.

To use the batch processing feature, first set hibernate.jdbc.batch\_size as batch size.

Eg: 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();

}

}

**Hibernate - Interceptors**

* Interceptor Interface methods are callbacks from the session to the application, allowing the application to inspect and/or manipulate properties of a persistent object before it is saved, updated, deleted or loaded.
* To build an interceptor you can either implement Interceptor class directly or extend EmptyInterceptor class.

**Hibernate Inheritance Mapping**

We can map the inheritance hierarchy classes with the table of the database.

There are three inheritance mapping strategies defined in the hibernate:

Eg scenario: Employee is either a contract or regular employee.

Another scenario: an interface Payment with the implementers CreditCardPayment, CashPayment, and ChequePayment.

**Table Per Hierarchy** - single table is required to map the whole hierarchy. we can map the whole hierarchy by single table only. Here, an extra column (also known as discriminator column) is created in the table to identify the class. Tag used: **<subclass>**

There is a limitation of this mapping strategy: columns declared by the subclasses cannot have NOT NULL constraints.

**Table Per Concrete class** - tables are created as per class. there will be three tables in the database having no relations to each other. Tag used: **<union-subclass>**

**Table Per Subclass** - tables are created as per class but related by foreign key. Here the tables are related to each other with primary and foreign key relationship. Tag used: **<joined-subclass>**

Hibernate does not support mixing <subclass>, <joined-subclass> and <union-subclass> mappings under the same root <class> element.

In addition, Hibernate supports a fourth, slightly different kind of polymorphism:

Implicit polymorphism:

Here the Employee interface may not be mentioned explicitly and also in both subclasses (ie) contract or regular employee classes should have the information which would have been given in Employee.

**Hibernate Versioning:**

* Once an object is saved in a database, we can modify that object any number of times and If we want to know how many no of times that an object is modified then we need to apply this versioning concept.
* When ever we use versioning then hibernate inserts version number as zero, when ever object is saved for the first time in the database.
* Later hibernate increments that version no by one automatically when ever a modification is done on that particular object.

we need the following two changes in our application for versioning

* Add one property of type int in our pojo class
* In hibernate mapping file, add an element called version soon after id element