## Hibernate Framework

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

## ORM Tool

An ORM tool simplifies the data creation, data manipulation and data access. It is a programming technique that maps the object to the data stored in the database.

The ORM tool internally uses the JDBC API to interact with the database.

## What is JPA?

Java Persistence API (JPA) is a Java specification that provides certain functionality and standard to ORM tools. The **javax.persistence** package contains the JPA classes and interfaces.

## Advantages of Hibernate Framework

### 1) Open Source and Lightweight

Hibernate framework is open source under the LGPL license and lightweight.

### 2) Fast Performance

The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled by default.

### 3) Database Independent Query

HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, if database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.

### 4) Automatic Table Creation

Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.

### 5) Simplifies Complex Join

Fetching data from multiple tables is easy in hibernate framework.

# Hibernate Architecture

The Hibernate architecture is categorized in four layers.

* Java application layer
* Hibernate framework layer
* Backhand api layer(JDBC API)
* Database layer

All the JPA annotations are defined in the **javax.persistence** package. Hibernate EntityManager implements the interfaces and life cycle defined by the JPA specification.

# Generator classes in Hibernate

The <generator> class is a sub-element of id. It is used to generate the unique identifier for the objects of persistent class. There are many generator classes defined in the Hibernate Framework.

1. assigned
2. increment
3. sequence
4. Native
5. hilo
6. Identity
7. uuid

### 1) assigned

It is the default generator strategy if there is no <generator> element . In this case, application assigns the id.

### 2) increment

It generates the unique id only if no other process is inserting data into this table. It generates **short**, **int** or **long** type identifier. If a table contains an identifier then the application considers its maximum value else the application consider that the first generated identifier is 1. For each attribute value, the hibernate increment the identifier by 1.

### 3) hilo

It uses high and low algorithm to generate the id of type short, int and long.

### 4) native

It uses identity, sequence or hilo depending on the database vendor.

### 5) uuid

It uses 128-bit UUID algorithm to generate the id. The returned id is of type String, unique within a network (because IP is used). The UUID is represented in hexadecimal digits, 32 in length.

# SQL Dialects in Hibernate

The dialect specifies the type of database used in hibernate so that hibernate generate appropriate type of SQL statements. For connecting any hibernate application with the database, it is required to provide the configuration of SQL dialect.

# 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:

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 nullable values are stored in the table .

1. @Entity
2. @Table(name = "employee101")
3. @Inheritance(strategy=InheritanceType.SINGLE\_TABLE)
4. @DiscriminatorColumn(name="type",discriminatorType=DiscriminatorType.STRING)
5. @DiscriminatorValue(value="employee")

#### 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.

In case of Table Per Concrete class, there will be three tables in the database having no relations to each other. There are two ways to map the table with table per concrete class strategy.

* By union-subclass element
* By self creating the table for each class

In case of Table Per Concrete class, tables are created per class. So there are no nullable values in the table. Disadvantage of this approach is that duplicate columns are created in the 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.

1. @Entity
2. @Table(name = "employee103")
3. @Inheritance(strategy=InheritanceType.JOINED)

### Collection Elements

The collection elements can have value or entity reference (another class object). We can use one of the 4 elements

* **element**
* **component-element**
* **one-to-many**, or
* **many-to-many**

The element and component-element are used for normal value such as string, int etc. whereas one-to-many and many-to-many are used to map entity reference.

#### List and Map are index based collection, so an extra column will be created in the table for indexing.

# if it stores value not entity reference that is why are going to use **element** instead of **one-to-many**.

If there is List<String>(list of values) then will use “element” if there is List<Object> then will use “one-to-many”.

# Mapping Bag in Collection Mapping

If our persistent class has List object, we can map the List by list or bag element in the mapping file. The bag is just like List but it doesn't require index element.

# Hibernate One to Many

If the persistent class has list object that contains the entity reference, we need to use one-to-many association to map the list element.

1. @OneToMany(cascade = CascadeType.ALL)
2. @JoinColumn(name="qid")
3. @OrderColumn(name="type")
4. **private** List<Answer> answers;

# Hibernate Many to Many

In this three tables will create.

1. @ManyToMany(targetEntity = Answer.**class**, cascade = { CascadeType.ALL })
2. @JoinTable(name = "q\_ans1123",
3. joinColumns = { @JoinColumn(name = "q\_id") },
4. inverseJoinColumns = { @JoinColumn(name = "ans\_id") })
5. **private** List<Answer> answers;

# Bidirectional Association

Bidirectional association allows us to fetch details of dependent object from both side. In such case, we have the reference of two classes in each other.

Let's take an example of Employee and Address, if Employee class has-a reference of Address and Address has a reference of Employee. Additionally, you have applied one-to-one or one-to-many relationship for the classes in mapping file as well, it is known as bidirectional association.

# Hibernate Lazy Collection

Lazy collection loads the child objects on demand, it is used to improve performance. Since Hibernate 3.0, lazy collection is enabled by default.

To use lazy collection, you may optionally use lazy="true" attribute in your collection. It is by default true, so you don't need to do this. If you set it to false, all the child objects will be loaded initially which will decrease performance in case of big data.

# Component Mapping

In component mapping, we will map the dependent object as a component. An component is an object that is stored as an value rather than entity reference. This is mainly used if the dependent object doen't have primary key. It is used in case of composition (HAS-A relation), that is why it is termed as component.

# Hibernate Transaction Management

The methods of Transaction interface are as follows:

1. **void begin()** starts a new transaction.
2. **void commit()** ends the unit of work unless we are in FlushMode.NEVER.
3. **void rollback()** forces this transaction to rollback.
4. **void setTimeout(int seconds)** it sets a transaction timeout for any transaction started by a subsequent call to begin on this instance.
5. **boolean isAlive()** checks if the transaction is still alive.
6. **void registerSynchronization(Synchronization s)** registers a user synchronization callback for this transaction.
7. **boolean wasCommited()** checks if the transaction is commited successfully.
8. **boolean wasRolledBack()** checks if the transaction is rolledback successfully.

# Hibernate Query Language (HQL)

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.

### Advantage of HQL

There are many advantages of HQL. They are as follows:

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

The query interface provides many methods. There is given commonly used methods:

1. **public int executeUpdate()** is used to execute the update or delete query.
2. **public List list()** returns the result of the ralation as a list.
3. **public Query setFirstResult(int rowno)** specifies the row number from where record will be retrieved.
4. **public Query setMaxResult(int rowno)** specifies the no. of records to be retrieved from the relation (table).
5. **public Query setParameter(int position, Object value)** it sets the value to the JDBC style query parameter.
6. **public Query setParameter(String name, Object value)** it sets the value to a named query parameter.

# HCQL (Hibernate Criteria Query Language)

The Hibernate Criteria Query Language (HCQL) is used to fetch the records based on the specific criteria. The Criteria interface provides methods to apply criteria such as retreiving all the records of table whose salary is greater than 50000 etc.

### Advantage of HCQL

The HCQL provides methods to add criteria, so it is **easy** for the java programmer to add criteria. The java programmer is able to add many criteria on a query.

The commonly used methods of Criteria interface are as follows:

1. **public Criteria add(Criterion c)** is used to add restrictions.
2. **public Criteria addOrder(Order o)** specifies ordering.
3. **public Criteria setFirstResult(int firstResult)** specifies the first number of record to be retreived.
4. **public Criteria setMaxResult(int totalResult)** specifies the total number of records to be retreived.
5. **public List list()** returns list containing object.
6. **public Criteria setProjection(Projection projection)** specifies the projection.

### Restrictions class

Restrictions class provides methods that can be used as Criterion. The commonly used methods of Restrictions class are as follows:

1. **public static SimpleExpression lt(String propertyName,Object value)** sets the **less than** constraint to the given property.
2. **public static SimpleExpression le(String propertyName,Object value)** sets the **less than or equal** constraint to the given property.
3. **public static SimpleExpression gt(String propertyName,Object value)** sets the **greater than** constraint to the given property.
4. **public static SimpleExpression ge(String propertyName,Object value)** sets the **greater than or equal** than constraint to the given property.
5. **public static SimpleExpression ne(String propertyName,Object value)** sets the **not equal** constraint to the given property.
6. **public static SimpleExpression eq(String propertyName,Object value)** sets the **equal** constraint to the given property.
7. **public static Criterion between(String propertyName, Object low, Object high)** sets the **between** constraint.
8. **public static SimpleExpression like(String propertyName, Object value)** sets the **like** constraint to the given property.

### Order class

The Order class represents an order. commonly used methods of Restrictions class :-

1. **public static Order asc(String propertyName)** applies the ascending order on the basis of given property.
2. **public static Order desc(String propertyName)** applies the descending order on the basis of given property.

### Example of HCQL to get the 10th to 20th record

1. Crietria c=session.createCriteria(Emp.**class**);
2. c.setFirstResult(10);
3. c.setMaxResult(20);
4. List list=c.list();

### Example of HCQL to get the records whose salary is greater than 10000

1. Crietria c=session.createCriteria(Emp.**class**);
2. c.add(Restrictions.gt("salary",10000));//salary is the propertyname
3. List list=c.list();

# Caching in Hibernate

Hibernate caching improves the performance of the application by pooling the object in the cache. It is useful when we have to fetch the same data multiple times.

There are mainly two types of caching:

* First Level Cache, and
* Second Level Cache

#### First Level Cache

Session object holds the first level cache data. It is enabled by default. The first level cache data will not be available to entire application. An application can use many session object.

#### Second Level Cache

SessionFactory object holds the second level cache data. The data stored in the second level cache will be available to entire application. But we need to enable it explicitely.

**Hibernate second level cache** uses *a common cache for all the session object of a session factory*. It is useful if you have multiple session objects from a session factory.

**SessionFactory** holds the second level cache data. It is global for all the session objects and not enabled by default.

Each implementation provides different cache usage functionality. There are four ways to use second level cache.

1. **read-only:** caching will work for read only operation.
2. **nonstrict-read-write:** caching will work for read and write but one at a time.
3. **read-write:** caching will work for read and write, can be used simultaneously.
4. **transactional:** caching will work for transaction.

Let's see the second level cache implementation and cache usage.

| **Implementation** | **read-only** | **nonstrict-read-write** | **read-write** | **transactional** |
| --- | --- | --- | --- | --- |
| EH Cache | Yes | Yes | Yes | No |
| OS Cache | Yes | Yes | Yes | No |
| Swarm Cache | Yes | Yes | No | No |
| JBoss Cache | No | No | No | Yes |

1. @Entity
2. @Table(name="emp1012")
3. @Cacheable
4. @Cache(usage=CacheConcurrencyStrategy.READ\_ONLY)
5. **public** **class** Employee {

### Hibernate JDBC Properties

| **Property** | **Description** |
| --- | --- |
| hibernate.connection.driver\_class | It represents the JDBC driver class. |
| hibernate.connection.url | It represents the JDBC URL. |
| hibernate.connection.username | It represents the database username. |
| hibernate.connection.password | It represents the database password. |
| Hibernate.connection.pool\_size | It represents the maximum number of connections available in the connection pool. |

## JPA vs. Hibernate

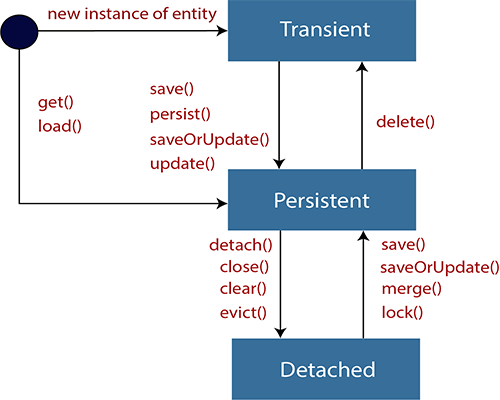
| **JPA** | **Hibernate** |
| --- | --- |
| Java Persistence API (JPA) defines the management of relational data in the Java applications. | Hibernate is an Object-Relational Mapping (ORM) tool which is used to save the state of Java object into the database. |
| It is just a specification. Various ORM tools implement it for data persistence. | It is one of the most frequently used JPA implementation. |
| It is defined in **javax.persistence** package. | It is defined in **org.hibernate** package. |
| The **EntityManagerFactory** interface is used to interact with the entity manager factory for the persistence unit. Thus, it provides an entity manager. | It uses **SessionFactory** interface to create Session instances. |
| It uses **EntityManager** interface to create, read, and delete operations for instances of mapped entity classes. This interface interacts with the persistence context. | It uses **Session** interface to create, read, and delete operations for instances of mapped entity classes. It behaves as a runtime interface between a Java application and Hibernate. |
| It uses **Java Persistence Query Language** (JPQL) as an object-oriented query language to perform database operations. | It uses **Hibernate Query Language** (HQL) as an object-oriented query language to perform database operations. |

# Hibernate Lifecycle

In Hibernate, either we create an object of an entity and save it into the database, or we fetch the data of an entity from the database. Here, each entity is associated with the lifecycle. The entity object passes through the different stages of the lifecycle.

The Hibernate lifecycle contains the following states: -

* Transient state
* Persistent state
* Detached state



### Transient state

* The transient state is the initial state of an object.
* Once we create an instance of POJO class, then the object entered in the transient state.
* Here, an object is not associated with the Session. So, the transient state is not related to any database.
* Hence, modifications in the data don't affect any changes in the database.
* The transient objects exist in the heap memory. They are independent of Hibernate.

### Persistent state

* As soon as the object associated with the Session, it entered in the persistent state.
* Hence, we can say that an object is in the persistence state when we save or persist it.
* Here, each object represents the row of the database table.
* So, modifications in the data make changes in the database.

### Detached State

* Once we either close the session or clear its cache, then the object entered into the detached state.
* As an object is no more associated with the Session, modifications in the data don't affect any changes in the database.
* However, the detached object still has a representation in the database.
* If we want to persist the changes made to a detached object, it is required to reattach the application to a valid Hibernate session.
* To associate the detached object with the new hibernate session, use any of these methods - load(), merge(), refresh(), update() or save() on a new session with the reference of the detached object.

# Difference between save() and persist() in HB

| **S** | **Key** | **save()** | **persist()** |
| --- | --- | --- | --- |
| 1 | Basic | It stores object in database | It also stores object in database |
| 2 | Return Type | It return generated id and return type is serializable | It does not return anything. Its void return type. |
| 3 | Transaction Boundaries | It can save object within boundaries and outside boundaries | It can only save object within the transaction boundaries |
| 4 | Detached Object | It will create a new row in the table for detached object | It will throw persistence exception for detached object |
| 5 | Supported by | It is only supported by Hibernate | It is also supported by JPA |

### 1) What is hibernate?

Hibernate is an open-source and lightweight ORM tool that is used to store, manipulate, and retrieve data from the database.

### 2) What is ORM?

ORM is an acronym for Object/Relational mapping. It is a programming strategy to map object with the data stored in the database. It simplifies data creation, data manipulation, and data access.

### 4) What are the core interfaces of Hibernate?

The core interfaces of Hibernate framework are:

* Configuration
* SessionFactory
* Session
* Query
* Criteria
* Transaction

### 5) Mention some of the advantages of using ORM over JDBC.

ORM has the following advantages over JDBC:

* Application development is fast.
* Management of transaction.
* Generates key automatically.
* Details of SQL queries are hidden.

### 6) Define criteria in terms of Hibernate.

The objects of criteria are used for the creation and execution of the object-oriented criteria queries.

### 8) List the key components of Hibernate.

Key components of Hibernate are:

* Configuration
* Session
* SessionFactory
* Criteria
* Query
* Transaction

### 9) Mention two components of Hibernate configuration object.

Database Connection

Class Mapping Setup

### 10) How is SQL query created in Hibernate?

The SQL query is created with the help of the following syntax:

Session.createSQLQuery

### 12) How is HQL query created?

The HQL query is created with the help of the following syntax:

Session.createQuery

### 14) Define persistent classes.

Classes whose objects are stored in a database table are called as persistent classes.

### 15) What is SessionFactory?

SessionFactory provides the instance of Session. It is a factory of Session. It holds the data of second level cache that is not enabled by default.

### 16) Is SessionFactory a thread-safe object?

Yes, SessionFactory is a thread-safe object, many threads cannot access it simultaneously.

### 17) What is Session?

It maintains a connection between the hibernate application and database.

It provides methods to store, update, delete or fetch data from the database such as persist(), update(), delete(), load(), get() etc.

### 18) Is Session a thread-safe object?

No, Session is not a thread-safe object, many threads can access it simultaneously. In other words, you can share it between threads.

### 20) What is the difference between get and load method?

The differences between get() and load() methods are given below.

| **N** | **get()** | **load()** |
| --- | --- | --- |
| 1 | Returns **null** if an object is not found. | Throws **ObjectNotFoundException** if an object is not found. |
| 2 | get() method always **hit the database**. | load() method **doesn't hit** the database. |
| 3 | It returns the real object, not the proxy. | It returns **proxy object.** |
| 4 | It should be used if **you are not sure** about the existence of instance. | It should be used if **you are sure** that instance exists. |

### 21) What is the difference between update and merge method?

The differences between update() and merge() methods are given below.

| **N** | **The update() method** | **merge() method** |
| --- | --- | --- |
| 1 | Update means to edit something. | Merge means to combine something. |
| 2 | update() should be used if the session doesn't contain an already persistent state with the same id. It means an update should be used inside the session only. After closing the session, it will throw the error. | merge() should be used if you don't know the state of the session, means you want to make the modification at any time. |

### 22) What are the states of the object in hibernate?

There are 3 states of the object (instance) in hibernate.

1. **Transient**: The object is in a transient state if it is just created but has no primary key (identifier) and not associated with a session.
2. **Persistent**: The object is in a persistent state if a session is open, and you just saved the instance in the database or retrieved the instance from the database.
3. **Detached**: The object is in a detached state if a session is closed. After detached state, the object comes to persistent state if you call lock() or update() method.

### 23) What are the inheritance mapping strategies?

There are 3 ways of inheritance mapping in hibernate.

1. Table per hierarchy
2. Table per concrete class
3. Table per subclass

### 24) How to make an immutable class in hibernate?

If you mark a class as mutable="false", the class will be treated as an immutable class. By default, it is mutable="true".

### 25) What is automatic dirty checking in hibernate?

The automatic dirty checking feature of Hibernate, calls update statement automatically on the objects that are modified in a transaction.

### 25) What is automatic dirty checking in hibernate?

The automatic dirty checking feature of Hibernate, calls update statement automatically on the objects that are modified in a transaction.

### 27) Is it possible to perform collection mapping with One-to-One and Many-to-One?

No, collection mapping can only be performed with One-to-Many and Many-to-Many.

### 28) What is lazy loading in hibernate?

Lazy loading in hibernate improves the performance. It loads the child objects on demand.

Since Hibernate 3, lazy loading is enabled by default, and you don't need to do lazy="true". It means not to load the child objects when the parent is loaded.

### 29) What is HQL (Hibernate Query Language)?

Hibernate Query Language is known as an object-oriented query language. It is like a structured query language (SQL).

The main advantage of HQL over SQL is:

1. You don't need to learn SQL
2. Database independent
3. Simple to write a query

### 30) What is the difference between first level cache and second level cache?

| **N** | **First Level Cache** | **Second Level Cache** |
| --- | --- | --- |
| 1 | First Level Cache is **associated with Session**. | Second Level Cache is associated with **SessionFactory**. |
| 2 | It is **enabled** by default. | It is **not enabled** by default. |