**Hibernate**

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

Object-Relational Mapping (ORM) is a technique that allows developers to map object-oriented code to a relational database system. It bridges the gap between the object-oriented programming paradigm used in application development and the relational database model used for data storage.



## What is JPA?

## JPA stands for Java Persistence API. It is a Java specification that defines a set of standards and interfaces for object-relational mapping (ORM) in Java applications. JPA provides a high-level, object-oriented approach to persisting data in relational databases.

## Advantages of Hibernate Framework

Following are the advantages of hibernate framework:

### **1) Open Source and Lightweight**

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

Open Source: Open source software refers to software that is released with a license that allows users to access, use, modify, and distribute the source code freely. The source code is made available to the public, fostering transparency, collaboration, and community-driven development. Open source projects often have vibrant communities, where developers contribute improvements, report issues, and share knowledge.

Lightweight: A lightweight software tool or framework is characterized by its small size, minimal resource requirements, and simplicity. Lightweight tools focus on providing essential features without excessive overhead or unnecessary complexity. They are designed to be efficient, easy to use, and performant.

### **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 includes many objects such as persistent object, session factory, transaction factory, connection factory, session, transaction etc.

This is the high level architecture of Hibernate with mapping file and configuration file.





A persistent object, in the context of software development, refers to an object that can be stored and retrieved from a persistent storage medium, such as a database, in a way that allows it to retain its state across different program executions.

#### **SessionFactory**

The SessionFactory is a factory of session and client of ConnectionProvider. It holds second level cache (optional) of data. The org.hibernate.SessionFactory interface provides factory method to get the object of Session.

#### **Session**

The session object provides an interface between the application and data stored in the database. It is a short-lived object and wraps the JDBC connection. It is factory of Transaction, Query and Criteria. It holds a first-level cache (mandatory) of data. The org.hibernate.Session interface provides methods to insert, update and delete the object. It also provides factory methods for Transaction, Query and Criteria.

#### **Transaction**

The transaction object specifies the atomic unit of work. It is optional. The org.hibernate.Transaction interface provides methods for transaction management.

#### **ConnectionProvider**

It is a factory of JDBC connections. It abstracts the application from DriverManager or DataSource. It is optional.

#### **TransactionFactory**

It is a factory of Transaction. It is optional.

# **Caching in Hibernate**

Caching in Hibernate refers to the mechanism of storing frequently accessed data in memory to improve the performance of database operations. Hibernate provides several levels of caching that can be utilized to reduce the number of database queries and improve overall application performance.

There are mainly two types of caching:

* First Level Cache, and
* Second Level Cache

#### **First Level Cache**

The first level cache, also known as the session cache, is enabled by default in Hibernate.

* It operates at the session or EntityManager level, storing objects that have been queried or persisted within a particular session.

The EntityManager is a core component of the Java Persistence API (JPA) and is responsible for managing entities, persisting data to the database, retrieving data, and executing queries. It serves as an interface between the application and the persistence context, which represents a set of managed entities.

* The first level cache is managed by Hibernate and is transparent to the developer.
* When an entity is fetched or loaded within a session, it is stored in the first level cache. Subsequent requests for the same entity within the same session will be retrieved from the cache, eliminating the need for additional database queries.

#### **Second Level Cache**

* The second level cache is an optional level of caching that operates at the session factory or EntityManagerFactory level.

The EntityManagerFactory is a central component in the Java Persistence API (JPA) that represents a factory for creating EntityManager instances. It is responsible for initializing the JPA provider, establishing the connection to the database, and managing the persistence units.

* Objects that are frequently accessed or shared among different sessions can be stored in the second level cache, reducing the need for repeated database queries.
* The second level cache can be configured for specific entities or collections, and it provides caching strategies such as read-only, transactional, and non-strict read-write.

In software development, an entity represents a specific object . It is typically a persistent data structure that corresponds to a database. Entities encapsulate both data and behavior, allowing developers to work with objects that have properties and methods associated with them.

A Java class can be easily transformed into an entity. For transformation the basic requirements are: -

* No-argument Constructor
* Annotation

### @Data annotation

The **@Data annotation** is equivalent to the combination of the following annotations:

* @Getter
* @Setter
* @RequiredArgsConstructor
* @ToString
* @EqualsAndHashCode

We can replace annotating a class with the annotations that are listed above and a single @Data annotation. The @Data annotation does the following work:

* It generates the getter methods for all the fields.
* It generates the setter methods for all the non-final fields.
* It generates the toString() method implementation.
* It generates appropriate equals() and hashCode() implementations, involving the fields of class.
* It generates a constructor that initializes all the final fields, as well as all the non-final fields with no initializer that have been marked with @NonNull, in order to ensure that the field is never null.

Java equals()

* The java equals() is a method of *lang.Object* class, and it is used to compare two objects.
* To compare two objects that whether they are the same, it compares the values of both the object's attributes.
* By default, two objects will be the same only if stored in the same memory location.

Java hashcode()

* A **hashcode** is an integer value associated with every object in Java, facilitating the hashing in hash tables.
* The hashcode() method returns the same hash value when called on two objects, which are equal according to the equals() method. And if the objects are unequal, it usually returns different hash values.

The @GeneratedValue annotation in JPA is used to specify the strategy for generating unique identifier values for entities. When using the @GeneratedValue annotation with the GenerationType.SEQUENCE strategy, the primary key values are generated using a database sequence.

# **Generator classes in Hibernate**

All the generator classes implements the **org.hibernate.id.IdentifierGenerator**[**interface**](https://www.javatpoint.com/interface-in-java). The application programmer may create one's own generator classes by implementing the IdentifierGenerator interface. Hibernate framework provides many built-in generator classes:

1. assigned
2. increment
3. sequence
4. hilo
5. native
6. identity
7. seqhilo
8. uuid
9. guid
10. select
11. foreign
12. sequence-identity

### **1) assigned**

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

### **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. Syntax

### **3) sequence**

It uses the sequence of the database. if there is no sequence defined, it creates a sequence automatically e.g. in case of Oracle database, it creates a sequence named HIBERNATE\_SEQUENCE.

### **4) hilo**

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

### **5) native**

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

### **6) identity**

It is used in Sybase, My SQL, MS SQL Server, DB2 and HypersonicSQL to support the id column. The returned id is of type short, int or long. It is responsibility of database to generate unique identifier.

### **7) seqhilo**

It uses high and low algorithm on the specified sequence name. The returned id is of type short, int or long.

### **8) 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.

### **9) guid**

|  |
| --- |
| It uses GUID generated by database of type string. It works on MS SQL Server and MySQL. |

### **10) select**

|  |
| --- |
| It uses the primary key returned by the database trigger. |

### **11) foreign**

|  |
| --- |
| It uses the id of another associated object, mostly used with <one-to-one> association. |

### **12) sequence-identity**

|  |
| --- |
| It uses a special sequence generation strategy. It is supported in Oracle 10g drivers only. |

# **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.

# **JPA Cascading Operations**

In JPA, if any operation is applied on an entity then it will perform on that particular entity only. These operations will not be applicable to the other entities that are related to it.

In the context of Hibernate, which is an object-relational mapping (ORM) framework, cascading refers to the automatic persistence operations that can be applied to associated entities when performing operations on a root entity. These persistence operations include saving, updating, deleting, and loading entities.