Hibernate:

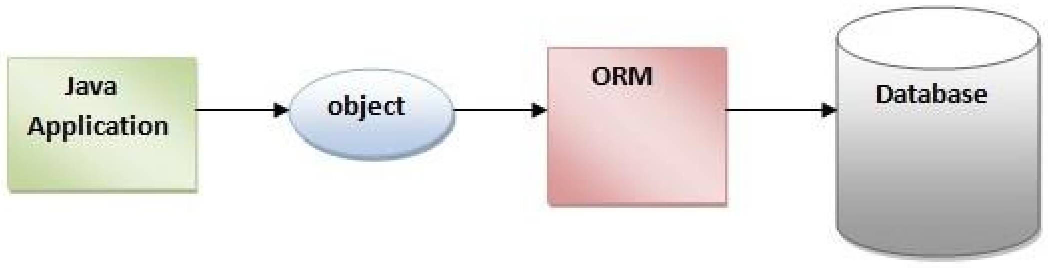
Hibernate Framework:

Hibernate is a Java framework that simplifies the development of Java applications to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. It was started in 2001 by Gavin King as an alternative to EJB2 style entity bean.

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:

Following are the 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 the database is changed for the project, we need to change the SQL query as well, which 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.

6) Provides Query Statistics and Database Status
Hibernate supports Query cache and provides statistics about query and database status.

Hibernate Architecture:

Hibernate is a framework which is used to develop persistence logic which is independent of Database software. In JDBC to develop persistence logic we deal with primitive types. Whereas Hibernate framework we use Objects to develop persistence logic which are independent of database software.

Hibernate Architecture:

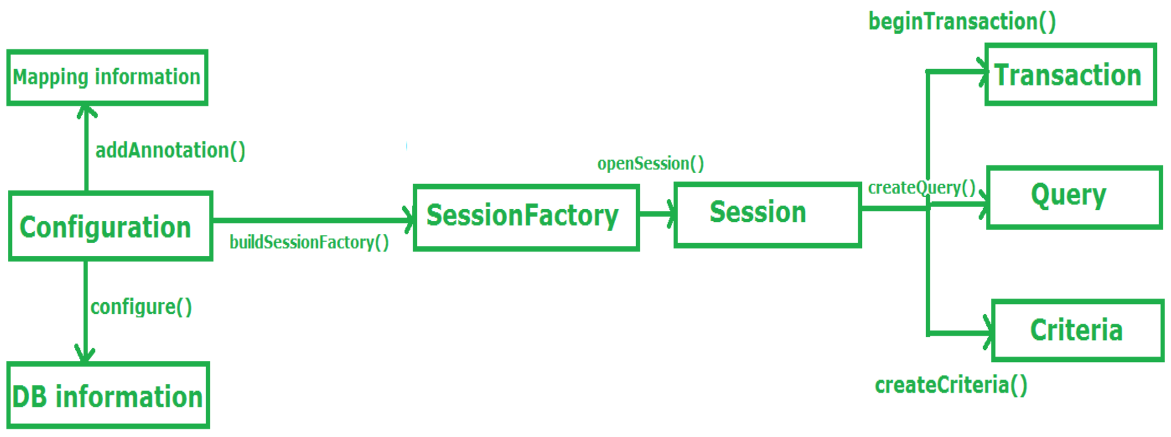


Fig: Hibernate Architecture

Configuration:

- Configuration is a class which is present in org.hibernate.cfg package. It activates the Hibernate framework. It reads both configuration files and mapping files.

It activate Hibernate Framework
Configuration cfg=new Configuration();
It read both cfg file and mapping files
cfg.configure();

- It checks whether the config file is syntactically correct or not.

- If the config file is not valid then it will throw an exception. If it is valid then it creates a meta-data in memory and returns the meta-data to the object to represent the config file.

SessionFactory:

- SessionFactory is an Interface which is present in org.hibernate package and it is used to create Session Object.
- It is immutable and thread-safe in nature.

buildSessionFactory() method gathers the meta-data which is in the cfg Object.
From the cfg object it takes the JDBC information and creates a JDBC Connection.
SessionFactory factory=cfg.buildSessionFactory();

Session:

- Session is an interface which is present in org.hibernate package. Session object is created based upon SessionFactory object i.e. factory.
- It opens the Connection/Session with Database software through Hibernate Framework.
- It is a light-weight object and it is not thread-safe.
- Session object is used to perform CRUD operations.

Session session = factory.openSession();

openSession() is a method provided by the SessionFactory that creates and returns a new Session instance. This session is not bound to any transaction or context and is independent of any ongoing transactions in the application.

We can also use getCurrentSession, which returns a Session bound to the current context, which is usually managed by a transaction manager or a framework like Spring.

Session session = sessionFactory.getCurrentSession();

Transaction:

- A Transaction object is used whenever we perform any operation and based upon that operation there is some change in the database.
- A Transaction object is used to give the instruction to the database to make the changes that happen because of operation as a permanent by using commit() method.

Transaction tx=session.beginTransaction();
tx.commit();

Query:

- Query is an interface that is present inside the org.hibernate package.
- A Query instance is obtained by calling Session.createQuery().

- This interface exposes some extra functionality beyond that provided by Session.iterate() and Session.find():
    1. A particular page of the result set may be selected by calling setMaxResults(), setFirstResult().
    2. Named query parameters may be used.

Criteria:

- Criteria is a simplified API for retrieving entities by composing Criterion objects.
- The Session is a factory for Criteria. Criterion instances are usually obtained via the factory methods on Restrictions.

☐ Flow of working during operation in Hibernate Framework: Suppose We want to insert an Object to the database. Here Object is nothing but persistence logic which we write on a java program and create an object of that program. If we want to insert that object in the database or we want to retrieve the object from the database. Now the question is how to hibernate, save the Object to the database or retrieve the object from the database. There are several layers through which the Hibernate framework goes to achieve the above task. Let us understand the layers/flow of Hibernate framework during performing operations:

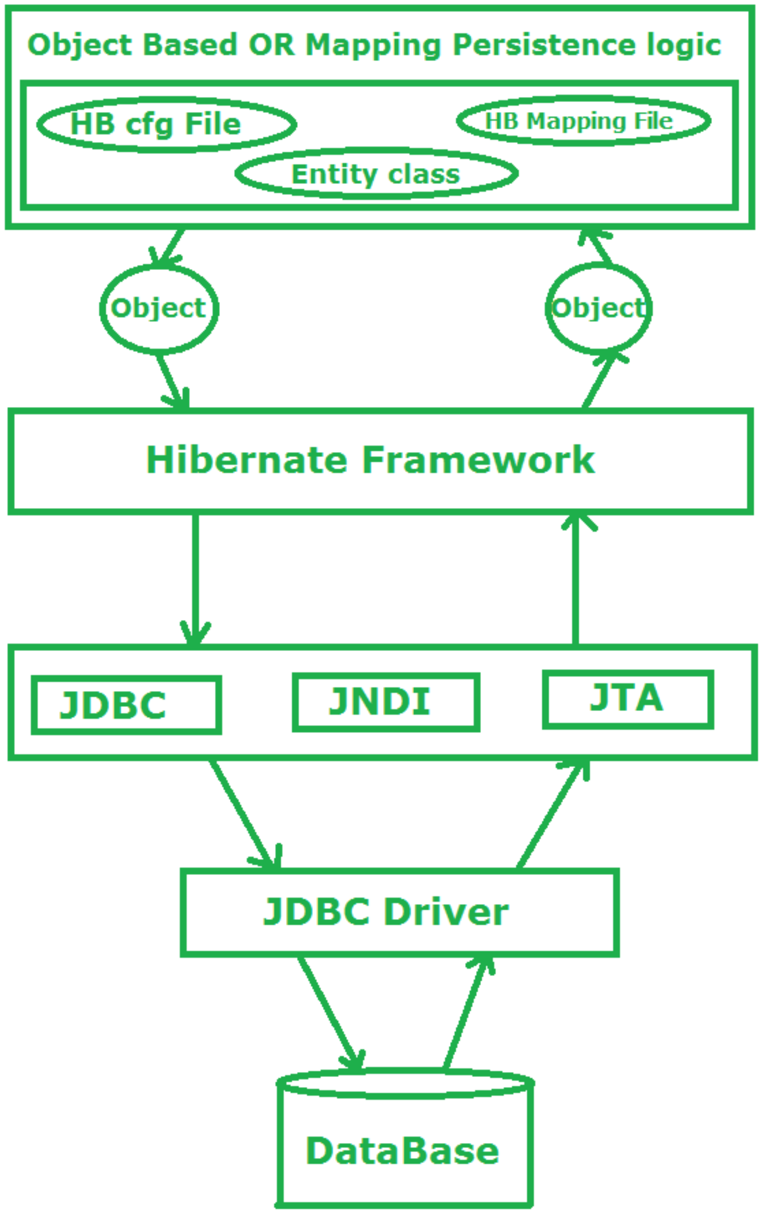


Fig: Working Flow of Hibernate framework to save/retrieve the data from the database in form of Object

Stage I: In the first stage, we will write the persistence logic to perform some specific operations to the database with the help of Hibernate Configuration file and Hibernate mapping file.

And after that we create an object of the particular class on which we wrote the persistence logic.

Stage II:In the second stage, our class which contains the persistence logic will interact with the hibernate framework where the hibernate framework gives some abstraction to perform some task. Now here the picture of java class is over.

Now Hibernate is responsible to perform the persistence logic with the help of layers which are below the Hibernate framework or we can say that the layers which are the internal implementation of Hibernate.

Stage III:In the third stage, our hibernate framework interacts with JDBC, JNDI, JTA etc to go to the database to perform that persistence logic.

Stage IV & V:In fourth & fifth stage, hibernate interacts with Database with the help of JDBC driver.

Now here hibernate performs that persistence logic which is nothing but CRUD operation.

If our persistence logic is to retrieve a record then in the reverse order it will display on the console of our java program in terms of Object.

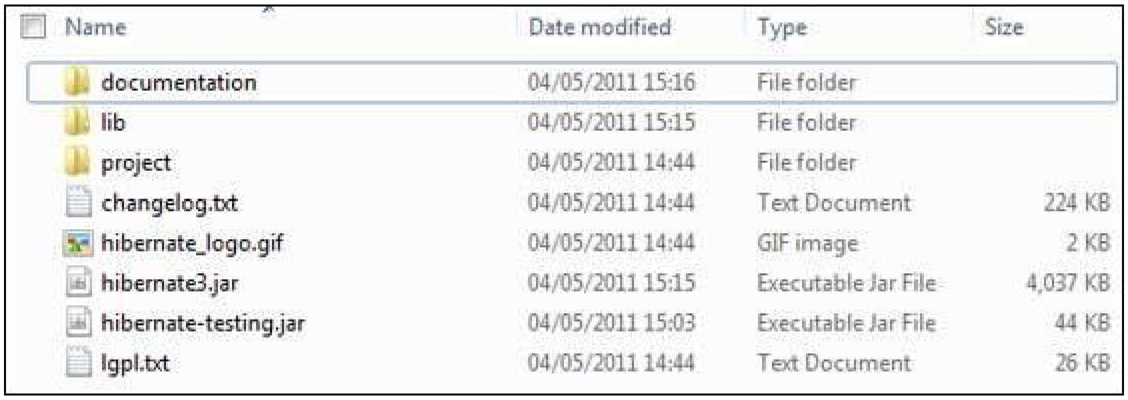
Installation and Configuration of Hibernate Framework:

Downloading Hibernate:

It is assumed that you already have the latest version of Java installed on your system. Following are the simple steps to download and install Hibernate on your system −

Make a choice whether you want to install Hibernate on Windows, or Unix and then proceed to the next step to download the .zip file for windows and .tz file for Unix.
Download the latest version of Hibernate from http://www.hibernate.org/downloads.

At the time of writing this tutorial, I downloaded hibernate-distribution 5.3.1. Final from mvnrepository and when you unzip the downloaded file, it will give you directory structure as shown in the following image



Name	Date modified	Type	Size
documentation	04/05/2011 15:16	File folder	
lib	04/05/2011 15:15	File folder	
project	04/05/2011 14:44	File folder	
changelog.txt	04/05/2011 14:44	Text Document	224 KB
hibernate_logo.gif	04/05/2011 14:44	GIF image	2 KB
hibernate3.jar	04/05/2011 15:15	Executable Jar File	4,037 KB
hibernate-testing.jar	04/05/2011 15:03	Executable Jar File	44 KB
lgpl.txt	04/05/2011 14:44	Text Document	26 KB

Installing Hibernate:

Once you downloaded and unzipped the latest version of the Hibernate Installation file, you need to perform the following two simple steps. Make sure you are setting your CLASSPATH variable properly otherwise you will face problems while compiling your application.

Now, copy all the library files from /lib into your CLASSPATH, and change your classpath variable to include all the JARs −
Finally, copy the hibernate3.jar file into your CLASSPATH. This file lies in the root directory of the installation and is the primary JAR that Hibernate needs to do its work.

Hibernate Prerequisites:

Following is the list of packages/libraries required by Hibernate and you should install them before starting with Hibernate. To install these packages, you will have to copy library files from /lib into your CLASSPATH, and change your CLASSPATH variable accordingly.

Sr.No.	Packages/Libraries
1	MySQL Connector/J MySQL Driver https://dev.mysql.com/downloads/connector/j/
2	Java EE Java EE API J2EE API

Following is the list of optional packages/libraries required by Hibernate and you can install them starting with Hibernate. To install these packages, you will have to copy library files from /lib into your CLASSPATH, and change your CLASSPATH variable accordingly.

Sr.No.	Packages/Libraries
1	dom4j XML parsing www.dom4j.org/
2	Xalan XSLT Processor https://xml.apache.org/xalan-j/
3	Xerces The Xerces Java Parser https://xml.apache.org/xerces-j/
4	cglib Appropriate changes to Java classes at runtime http://cglib.sourceforge.net/
5	log4j Logging Faremwork https://logging.apache.org/log4j
6	Commons Logging, Email etc. https://jakarta.apache.org/commons

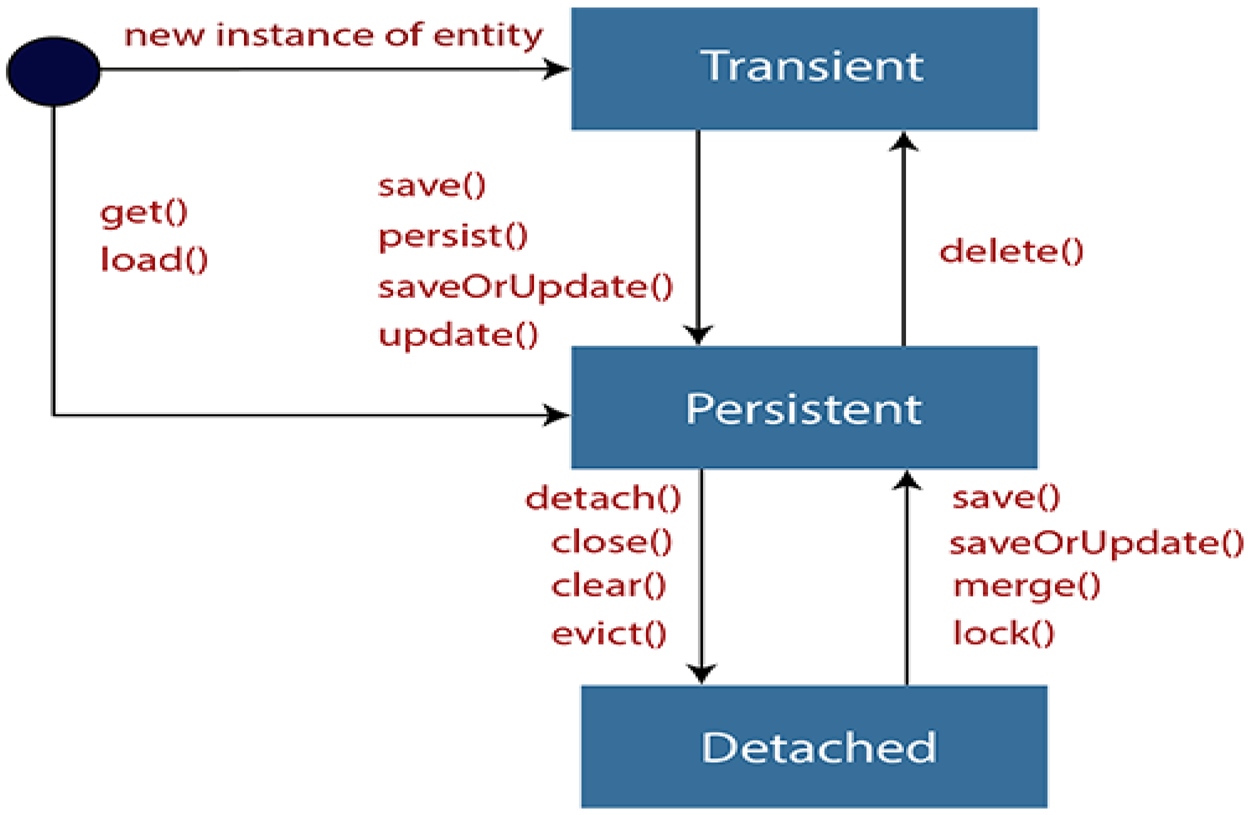
7	SLF4J Logging Facade for Java https://www.slf4j.org

Java Objects in Hibernate: 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 life cycle.

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.

```
Employee e=new Employee(); //Here, object enters in the transient state.
e.setId(101);
e.setFirstName("Gaurav");
e.setLastName("Chawla");
```

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.

We can use any of the following methods for the persistent state.

1. session.save(e); can be used to save entities to the database.
2. session.persist(e); adds an entity object to the persistent context, which tracks any future changes.
3.session.update(e); is used in ORM frameworks like Hibernate to update an existing entity in the database.
session.saveOrUpdate(e); The method session.saveOrUpdate(e); is used in ORM frameworks like Hibernate to either save a new entity or update an existing one, depending on the state of the entity e.
session.lock(e); The lock() method allows you to specify a lock mode, which controls how the entity can be accessed by other transactions.
session.merge(e); used to update the state of an entity e from a detached state to a managed state. It’s particularly useful in scenarios where you have an entity that may have been modified outside of the current session.

Detached State:

- Once we either close the session or clear its cache, then the object enters 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.

We can use any of the following methods for the detached state.

session.close(); is used in Hibernate to close the current session.
session.clear(); in Hibernate is used to clear the session of all entities that it is currently managing.
session.detach(e); in Hibernate is used to detach a specific entity e from the current

session.
session.evict(e); in Hibernate is used to remove a specific entity e from the session's context

Inheritance Mapping:

Inheritance mapping is a technique used in object-relational mapping (ORM) to handle class hierarchies in relational databases. It helps to define how a parent class and its subclasses are represented in the database. There are several strategies for inheritance mapping:

1. Single Table Inheritance:
   - All classes in the hierarchy are stored in a single table.
   - A discriminator column is used to differentiate between the different types.
   - Pros: Simplicity and performance; fewer joins.
   - Cons: Can lead to a lot of nullable columns if subclasses have different attributes.
2. Class Table Inheritance:
   - Each class has its own table, and the table for a subclass contains only the fields specific to that subclass.
   - The parent class's table holds the common attributes.
   - Pros: Normalization; no nullable columns.
   - Cons: More complex queries due to joins.
3. Concrete Table Inheritance:
   - Each class in the hierarchy is represented by its own table, with no parent class table.
   - Each table contains all the fields needed for that class.
   - Pros: Simple queries, no joins required.
   - Cons: Data duplication; changes to the parent class must be manually propagated.
4. Mapped Superclass:
   - A superclass can be defined that is not directly mapped to a table but provides shared fields to subclasses.
   - Useful for code reuse without creating a separate table for the superclass.

Collection Mapping in hibernate:

In Hibernate, collection mapping allows you to manage and persist collections of entities or basic types using various strategies. Here’s an overview of how to map collections in Hibernate, including common annotations and configurations:

1. One-to-Many Relationships:

- Mapping: Use the @OneToMany annotation in the parent entity to define the relationship.

Example:

```
@Entity
public class Customer {
  @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL, orphanRemoval = true)
    private List<Order> orders = new ArrayList<>();
}

@Entity
public class Order {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;
}
```

2. Many-to-Many Relationships:

- Mapping: Use the @ManyToMany annotation with a join table.

Example:

```
@Entity
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToMany
    @JoinTable(
        name = "student_course",
        joinColumns = @JoinColumn(name = "student_id"),
        inverseJoinColumns = @JoinColumn(name = "course_id")
    )
    private Set<Course> courses = new HashSet<>();
}

@Entity
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ManyToMany(mappedBy = "courses")
    private Set<Student> students = new HashSet<>();
}
```

3. Element Collection:

- Mapping: Use the @ElementCollection annotation for collections of basic types or embeddable classes.

Example:

```
@Entity
public class User {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @ElementCollection
    @CollectionTable(name = "user_phones", joinColumns = @JoinColumn(name = "user_id"))
    @Column(name = "phone_number")
    private List<String> phoneNumbers = new ArrayList<>();
}
```

4. Embedded Collections:

- Mapping: For collections of embeddable types, use @ElementCollection with @Embeddable.

Example:

```
@Embeddable
public class Address {
    private String street;
    private String city;
}

@Entity
public class Person {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
```

```
   @ElementCollection
       @CollectionTable(name = "person_addresses", joinColumns = @JoinColumn(name =
"person_id"))
   private List<Address> addresses = new ArrayList<>();
}
```

5. Collection Types:

- Lists: Use List with @OrderColumn if you need to maintain the order.
- Sets: Use Set to ensure uniqueness.
- Maps: Use Map for key-value pairs.

Hibernate Query Language:

Hibernate is a Java framework that makes it easier to create database-interactive Java applications. In HQL, instead of a table name, it uses a class name. As a result, it is a query language that is database-independent.

Hibernate converts HQL queries into SQL queries, which are used to perform database actions. Although Native SQL may be used directly with Hibernate, it is encouraged to utilize HQL wherever feasible to prevent database portability issues.

HQL has many benefits. Some benefits are:

- HQL is database-independent.
- polymorphic queries supported which are type-safe.
- It is portable and easy to learn for Java programmers.

The **Query interface** provides object-oriented methods and capabilities for representing and manipulating HQL queries.

Hibernate Query Language (HQL) Clauses:
There are many HQL clauses available to interact with relational databases, and several of them are listed below:

1. FROM Clause
2. SELECT Clause
3. WHERE Clause
4. ORDER BY Clause
5. UPDATE Clause
6. DELETE Clause
7. INSERT Clause

For details and examples regarding each clause is mentioned below.

1. FROM Clause:

To load a whole persistent object into memory, the FROM clause is used.

String hib = "**FROM** Student";

Query query = session.createQuery(hib);

List results = query.list();

2. SELECT Clause:

The SELECT clause is used when only a *few attributes of an object* are required rather than the entire object.

String hib = "**SELECT** S.roll FROM Student S";

Query query = session.createQuery(hib);

List results = query.list();

3. WHERE Clause:

Filtering records is done with the WHERE clause. It's used to *retrieve only the records that meet a set of criteria*.

String hib = "FROM Student S **WHERE** S.id = 5";

Query query = session.createQuery(hib);

List results = query.list();

4. ORDER BY Clause:

The ORDER BY clause is used to *sort the results* of an HQL query.

String hib = "FROM Student S WHERE S.id > 5 **ORDER BY** S.id **DESC**";

Query query = session.createQuery(hib);

List results = query.list();

1. ***Order By – DESC*** *will sort in descending order*

2. ***Order By – ASC*** *will sort in ascending order*

5. UPDATE Clause

The UPDATE clause is required to *update the value* of an attribute.

```
String hib = "UPDATE Student set name=:n WHERE roll=:i";

Query q=session.createQuery(hib);
q.setParameter("n","John");
q.setParameter("i",23);

int status=q.executeUpdate();
System.out.println(status);
```

6. DELETE Clause

It is required to *delete a value* of an attribute.

```
String hib = "DELETE FROM Student WHERE id=10";

Query query=session.createQuery(hib);
query.executeUpdate();
```

7. INSERT Clause

It is required to *Insert values* into the relation.

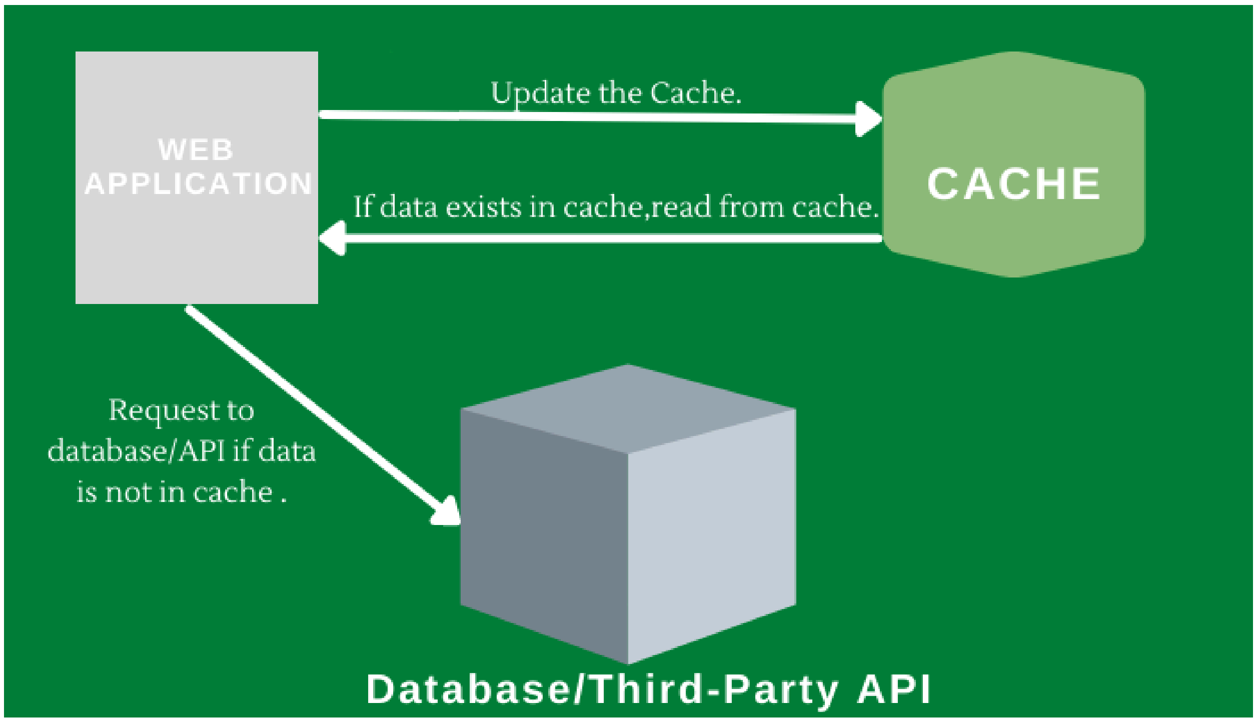
```
String hib = "INSERT INTO Student(first_name, last_name)" +
            "SELECT first_name, last_name FROM backup_student";

Query query = session.createQuery(hib);
int result = query.executeUpdate();
```

<u>Spring Boot – Caching:</u>

Spring Boot is a project that is built on top of the Spring Framework that provides an easier and faster way to set up, configure, and run both simple and web-based applications. It is one of the popular frameworks among developers these days because of its rapid production-ready environment which enables the developers to directly focus on the logic instead of struggling with the configuration and setup. It is a microservice-based framework used to create a stand-alone Spring-based application that we can run with minimal Spring configurations.

Salient Features

- There is no requirement for heavy **XML** configuration.
- Developing and testing the Spring Boot application is easy as it offers a **CLI** based tool and it also has embedded HTTP servers such as Tomcat, Jetty, etc.
- Since it uses convention over configuration, it increases productivity and reduces development time.


WEB APPLICATION
Update the Cache.
CACHE
If data exists in cache,read from cache.
Request to database/API if data is not in cache .
Database/Third-Party API

Clarification on Database Caching.

A **Cache** is any temporary storage location that lies between the application and persistence database or a third-party application that stores the most frequently or recently accessed data so that future requests for that data can be served faster. It increases data retrieval performance by reducing the need to access the underlying slower storage layer. Data access from memory is always faster in comparison to fetching data from the database. Caching keeps frequently accessed objects, images, and data closer to where you need them, speeding up access by not hitting the database or any third-party application multiple times for the same data and saving monetary costs. Data that does not change frequently can be cached.

Types of Caching:

There are mainly 4 types of Caching :

1. CDN Caching
2. Database Caching
3. In-Memory Caching
4. Web server Caching

1. CDN Caching

A **content delivery network (CDN)** is a group of distributed servers that speed up the delivery of web content by bringing it closer to where users are. Data centers across the globe use caching, to deliver internet content to a web-enabled device or browser more quickly through a server near you, reducing the load on an application origin and improving the user experience. CDNs cache content like web pages, images, and video in proxy servers near your physical location.

2. Database Caching

Database caching improves scalability by distributing query workload from the backend to multiple front-end systems. It allows flexibility in the processing of data. It can significantly reduce latency and increase throughput for read-heavy application workloads by avoiding querying a database too much.

3. In-Memory Caching

In-Memory Caching increases the performance of the application. An in-memory cache is a common query store, therefore, relieves databases of reading workloads. Redis cache is one of the examples of an in-memory cache. Redis is distributed, an advanced caching tool that allows backup and restore facilities. In-memory Cache provides query functionality on top of caching.

4. Web server Caching

Web server caching stores data, such as a copy of a web page served by a web server. It is cached or stored the first time a user visits the page and when the next time a user requests the same page, the content will be delivered from the cache, which helps keep the origin server from getting overloaded. It enhances page delivery speed significantly and reduces the work needed to be done by the backend server.

Introduction to Spring Integration:

Spring Integration is an open source framework for enterprise application integration. It is a lightweight framework that builds upon the core Spring framework. It is designed to enable the development of integration solutions typical of event-driven architectures and messaging-centric architectures.

Spring Integration extends the Spring programming model to support the well-known Enterprise Integration Patterns. Enables lightweight messaging within Spring-based applications and supports integration with external systems via declarative adapters. Those adapters provide a higher-level of abstraction over Spring's support for remoting, messaging, and scheduling.

Using the Spring Framework encourages developers to code using interfaces and use dependency injection (DI) to provide a Plain Old Java Object (POJO) with the dependencies it needs to perform its tasks. Spring Integration takes this concept one step further, where POJOs are wired together using a messaging paradigm and individual components may not be aware of other components in the application. Such an application is built by assembling fine-grained reusable components to form a higher level of functionality. With careful design, these flows can be modularized and also reused at an even higher level.

In addition to wiring together fine-grained components, Spring Integration provides a wide selection of channel adapters and gateways to communicate with external systems. Channel Adapters are used for one-way integration (send or receive); gateways are used for request/reply scenarios (inbound or outbound). For a full list of adapters and gateways, refer to the reference documentation.

The Spring Cloud Stream project builds on Spring Integration, where Spring Integration is used as an engine for message-driven microservices.