Hibernate

What is Hibernate?

Hibernate is an **Object-Relational Mapping (ORM)** framework for Java that simplifies interaction between Java applications and relational databases. It maps Java classes to database tables and Java data types to SQL data types.

Hibernate Architecture Overview

1. **Configuration** – Reads hibernate.cfg.xml or .properties files.
2. **SessionFactory** – A factory for Session objects; created once during application startup.
3. **Session** – A single-threaded, short-lived object representing a unit of work with the database.
4. **Transaction** – Manages atomic units of work.
5. **Query / Criteria** – Used to retrieve data from the database.
6. **Entity** – A POJO (Plain Old Java Object) annotated with @Entity.

Hibernate Setup Steps

Step 1:Add Hibernate Dependencies(In Maven project):

```
  <project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

  <modelVersion>4.0.0</modelVersion>

  <groupId>com.tap</groupId>

  <artifactId>Maven</artifactId>

  <version>0.0.1-SNAPSHOT</version>

  <dependencies>

<!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->

        <dependency>
          <groupId>com.mysql</groupId>
          <artifactId>mysql-connector-j</artifactId>
          <version>9.2.0</version>
        </dependency>

        <!-- https://mvnrepository.com/artifact/org.hibernate.orm/hibernate-core -->
        <dependency>
          <groupId>org.hibernate.orm</groupId>
          <artifactId>hibernate-core</artifactId>
          <version>6.6.2.Final</version>
        </dependency>
</dependencies>
  </project>
```

Note: Use <Dependency>use **hibernate** and **mySQL** jar files </Dependency> is inside the <Dependencies> Tag

Website: Maven Repository

Step 2: Create hibernate.cfg.xml (or) Settings.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<hibernate-configuration>
  <session-factory>
     <property name="connection.driver_class">com.mysql.cj.jdbc.Driver</property>
    <property name="dialect">org.hibernate.dialect.MySQL8Dialect</property>
    <property name="connection.url">jdbc:mysql://localhost:3306/hibernate</property>
    <property name="connection.username">root</property>
    <property name="connection.password">root</property>
    <property name="hibernate.show_sql">true</property>
```

```
 </session-factory>

</hibernate-configuration>
```

Step 3: Create POJO (Entity Class)

```
import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name="student")

public class Student

{

        @Id

        @Column(name="id")

        int id;

        @Column(name="name")

        String name;

        @Column(name="email")

        String email;

        public Student() {

        }

  public Student(int id, String name, String email) {

                super();

                this.id = id;

                this.name = name;

                this.email = email;

        }

      public int getId() {
```

```
		return id;
	}

	public void setId(int id) {
		this.id = id;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getEmail() {
		return email;
	}
	 public void setEmail(String email) {
		this.email = email;
	}
	@Override
	public String toString() {
		return id+" "+name+" "+email;
	}
}
```

Step 5: CRUD Operations

1.Create(insert)

Student student=**new** Student(6,"suheb","suheb19@gmail.com");

session.persist(student);

transaction.commit();

System.out.println("row inserted");

2.Read (Retrieve)

***GET student**

Student student = session.get(Student.class,6);

System.out.println(student.getId()+" "+student.getName()+" "+student.getEmail());

3.Update

Student student=session.get(Student.class,2);

student.setEmail("sathwwik556@gmail.com");

session.update(student);

transaction.commit();

System.out.println(student);

4.Delete

Student student=session.get(Student.class,4);

session.delete(student);

transaction.commit();

5. GET ALL students

// SQL-->select * from student;

// HQL-->From student s

Query q=session.createQuery("From student s");

```
List <Student> student=q.getResultList();
for (Student s : student) {
    System.out.println(s);
}
```

6.Update Query

```
//SQL-->UPDATE employee SET salary=salary=1000 where
salary>=70000;'

//HQL-->UPDATE employee e set e.salary=e.salary+1000
e.salary>=70000;

String update="UPDATE Employee e set e.salary = e.salary+2000 WHERE
e.salary>=50000";
Query query=session.createQuery(update);
query.executeUpdate();
transaction.commit();
System.out.println("salary updated");
```

Sample Code:

```
import java.util.List;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.Transaction;
import org.hibernate.cfg.Configuration;
import org.hibernate.query.Query;
public class App {
        public static void main(String[] args) {
            Configuration config=new Configuration();
```

```
System.out.println(config);
config.configure("setting.xml");
config.addAnnotatedClass(Student.class);
SessionFactory factory =config.buildSessionFactory();
Session session = factory.openSession();
Transaction transaction=session.beginTransaction();
// 1.INSERT student:
 Student student=new Student(6,"suheb","suheb19@gmail.com");
 session.persist(student);
 transaction.commit();
System.out.println("row inserted");
  // 2.GET student
 Student student = session.get(Student.class,6);
 System.out.println(student.getId()+" "+student.getName()+"
"+student.getEmail());
// 3.UPDATE
Student student=session.get(Student.class,2);
student.setEmail("sathwwik556@gmail.com");
session.update(student);
transaction.commit();
System.out.println(student);
// 4.DELETE
Student student=session.get(Student.class,4);
session.delete(student);
transaction.commit();
```

```
        // 5. GET ALL students
        // SQL-->select * from student;
        // HQL-->From student s
        Query q=session.createQuery("From student s");
        List <Student> student=q.getResultList();
        for (Student s : student) {
            System.out.println(s);
        }
    }
}
```

➔ **Update All Employee’s email change with Single Query**

```
//          SQL---->UPDATE Employee SET email=?;
//          HQL---->UPDATE Employee e SET e.email=?1;

            String HQL="UPDATE Employee e SET e.email=?1";
            MutationQuery query=session.createMutationQuery(HQL);
            query.setParameter(1,"tap503@gmail.com");
            query.executeUpdate();
            transaction.commit();
```

Hibernate Life Cycle

🔄 Hibernate Object Lifecycle Overview

In Hibernate, an object (usually a Java POJO) goes through **four main states**:

State	Description	Hibernate Session Required	Persistent in DB
Transient	Object is just created using `new`, not associated with Hibernate session.	❌	❌
Persistent	Object is associated with a session and will be saved/updated in DB.	✅	✅ (after flush)
Detached	Object was persistent, but session is closed.	❌	✅
Removed	Object is marked for deletion; will be deleted from DB.	✅	❌ (after commit)

📊 Hibernate Lifecycle Table

State	How Object Enters This State	Behavior
Transient	`new Student()`	No DB interaction, not tracked by Hibernate.
Persistent	`session.save(obj)`, `session.persist(obj)`, or `get()`	Hibernate tracks changes; auto-synced to DB on commit/flush.
Detached	After `session.close()` or `evict(obj)`	Object still exists in JVM, but not managed by Hibernate.
Removed	`session.delete(obj)`	Object will be removed from DB on commit or flush.