**Hibernate: A Comprehensive Guide (Basics to Advanced)**

**1. Introduction to Hibernate**

**What is Hibernate?**

Hibernate is an **Object-Relational Mapping (ORM) framework** for Java applications. It allows developers to interact with relational databases using Java objects instead of writing SQL queries manually.

**Why Use Hibernate?**

* **Eliminates JDBC Boilerplate Code**: Reduces repetitive database access code.
* **Automatic Table Mapping**: Maps Java objects to database tables.
* **Database Independence**: Works with multiple databases (MySQL, PostgreSQL, Oracle, etc.).
* **Caching Mechanism**: Enhances performance by reducing database hits.
* **HQL (Hibernate Query Language)**: An object-oriented way to query the database.
* **Transaction Management**: Ensures data consistency.

**2. Hibernate Architecture**

**Core Components**

1. **Configuration** - Defines database connection settings and Hibernate properties.
2. **SessionFactory** - Creates Session objects and is a heavyweight object, usually created once per application.
3. **Session** - A lightweight object representing a single unit of work (CRUD operations).
4. **Transaction** - Manages commit/rollback operations.
5. **HQL (Hibernate Query Language)** - Used for querying objects instead of raw SQL.

**Hibernate Workflow**

1. Load hibernate.cfg.xml configuration.
2. Create a SessionFactory.
3. Obtain a Session from SessionFactory.
4. Perform CRUD operations using Session.
5. Commit/rollback transactions.
6. Close the session to release resources.

**3. Hibernate Setup & Configuration**

**Maven Dependencies**

To use Hibernate in a Java project, add the following dependencies in pom.xml:

<dependencies>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>6.2.0.Final</version>

</dependency>

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>8.0.33</version>

</dependency>

</dependencies>

**Hibernate Configuration (hibernate.cfg.xml)**

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/your\_db</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">password</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.hbm2ddl.auto">update</property>

<property name="hibernate.show\_sql">true</property>

</session-factory>

</hibernate-configuration>

**SessionFactory Creation**

SessionFactory factory = new Configuration().configure("hibernate.cfg.xml").buildSessionFactory();

Session session = factory.openSession();

**4. Hibernate Annotations & Entity Mapping**

**Creating an Entity Class**

import jakarta.persistence.\*;

@Entity

@Table(name = "users")

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "username", nullable = false, unique = true)

private String username;

@Column(name = "email")

private String email;

// Getters & Setters

}

**Performing CRUD Operations**

**Create (Insert Data)**

Session session = factory.openSession();

Transaction tx = session.beginTransaction();

User user = new User();

user.setUsername("john\_doe");

user.setEmail("john@example.com");

session.save(user);

tx.commit();

session.close();

**Read Data**

User user = session.get(User.class, 1);

System.out.println(user.getUsername());

**Update Data**

User user = session.get(User.class, 1);

user.setEmail("newemail@example.com");

session.update(user);

tx.commit();

**Delete Data**

User user = session.get(User.class, 1);

session.delete(user);

tx.commit();

**5. Hibernate Query Language (HQL)**

HQL allows queries using entity names instead of table names.

**Basic Queries**

**Fetch All Users**

Query query = session.createQuery("FROM User");

List<User> users = query.list();

**Fetch User by Name**

Query query = session.createQuery("FROM User WHERE username = :name");

query.setParameter("name", "john\_doe");

User user = (User) query.uniqueResult();

**Update Email**

Query query = session.createQuery("UPDATE User SET email = :email WHERE id = :id");

query.setParameter("email", "newemail@example.com");

query.setParameter("id", 1);

query.executeUpdate();

**6. Hibernate Relationships**

**One-To-One Mapping**

@Entity

public class UserProfile {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@OneToOne

@JoinColumn(name = "user\_id")

private User user;

}

**One-To-Many Mapping**

@Entity

public class User {

@OneToMany(mappedBy = "user", cascade = CascadeType.ALL)

private List<Order> orders;

}

**Many-To-Many Mapping**

@Entity

public class Student {

@ManyToMany

@JoinTable(

name = "student\_course",

joinColumns = @JoinColumn(name = "student\_id"),

inverseJoinColumns = @JoinColumn(name = "course\_id")

)

private List<Course> courses;

}

**7. Hibernate Caching**

**First-Level Cache (Default)**

* Works within a single session.
* Reduces redundant queries.

**Second-Level Cache**

* Shared across sessions.
* Uses EhCache, Redis, etc.

**Enabling EhCache**

<property name="hibernate.cache.use\_second\_level\_cache">true</property>

<property name="hibernate.cache.region.factory\_class">org.hibernate.cache.ehcache.EhCacheRegionFactory</property>

**8. Hibernate with Spring Boot**

Hibernate can be easily integrated with Spring Boot using spring-boot-starter-data-jpa.

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>