**Hibernate**

1. **What is JPA, and how is it different from Hibernate?**

JPA (Java Persistence API) is a specification for ORM (Object-Relational Mapping) in Java. It defines a set of interfaces and guidelines for persisting Java objects to relational databases.

Hibernate is an implementation of JPA. It provides additional features beyond JPA, such as caching, lazy loading, and improved query capabilities.

1. **What are the key features of Hibernate?**
   1. **ORM (Object-Relational Mapping)** – Maps Java objects to database tables.
   2. HQL (Hibernate Query Language) **– More powerful than SQL, allows object-based queries.**
   3. Caching **– Supports first-level and second-level caching to improve performance.**
   4. Lazy and Eager Loading **– Controls how entity relationships are fetched.**
   5. Automatic **Schema** Generation **– Can create, update, and validate database schemas.**
2. **Explain the difference between @Entity, @Table, and @Column.**
   1. @Entity – Marks a Java class as a JPA entity (mapped to a database table).
   2. @Table(name = "table\_name") – Specifies the table name for the entity (if different from the class name).
   3. @Column(name = "column\_name") – Defines the mapping between a field and a database column.
   4. Ex.

**@Entity**

**@Table(name = "users")**

**public class User {**

**@Id**

**@GeneratedValue(strategy = GenerationType.IDENTITY)**

**private Long id;**

**@Column(name = "user\_name", nullable = false, unique = true)**

**private String username;**

**}**

1. **What are the different fetching strategies in Hibernate?**

Hibernate provides two fetching strategies:

* 1. **Eager Fetching** (FetchType.EAGER) – Loads related entities immediately.
  2. **Lazy Fetching (FetchType.LAZY) – Loads related entities only when accessed**.

Ex.

@OneToMany(mappedBy = "user", fetch = FetchType.LAZY)

private List<Order> orders;

1. **What is the difference between Session and EntityManager?**

|  |  |  |
| --- | --- | --- |
| **Feature** | **Session (Hibernate)** | **EntityManager (JPA)** |
| API | Hibernate API | JPA API |
| Caching | First-level cache | First-level cache |
| Query Language | HQL (Hibernate Query Language) | JPQL (Java Persistence Query Language) |
| Transaction Handling | Managed by Hibernate | Managed by JPA |

1. **Explain the different types of caches in Hibernate.**

Answer:

1. **First-Level Cache** – Enabled by default, associated with a Session.
2. **Second-Level Cache – Shared across multiple sessions, requires configuration (Ehcache, Redis, etc.).**
3. **Query Cache – Caches the results of queries to improve performance.**

Example of Second-Level Cache Configuration:

properties

hibernate.cache.use\_second\_level\_cache=true

hibernate.cache.region.factory\_class=org.hibernate.cache.ehcache.EhCacheRegionFactory

1. **What are the different inheritance strategies in JPA?**

JPA provides three inheritance strategies:

1. Single Table (@Inheritance(strategy = InheritanceType.SINGLE\_TABLE))
   1. Uses a single table for all entities in the hierarchy.
2. Joined Table (@Inheritance(strategy = InheritanceType.JOINED))
   1. Creates separate tables for each entity, maintaining relationships.
3. Table Per Class (@Inheritance(strategy = InheritanceType.TABLE\_PER\_CLASS))
   1. Each subclass has its own table.
   2. ex

@Entity

@Inheritance(strategy = InheritanceType.SINGLE\_TABLE)

@DiscriminatorColumn(name = "type")

public class Vehicle { }

@Entity

@DiscriminatorValue("Car")

public class Car extends Vehicle { }

1. **How does Hibernate handle transactions?**

Transactions in Hibernate are managed using @Transactional in Spring or manually with beginTransaction() and commit().

Ex.

**@Transactional**

**public void saveUser(User user) {**

**entityManager.persist(user);**

**}**

1. **What is the N+1 Query Problem, and how can you avoid it?**

**N+1 Problem:** When fetching a collection, Hibernate runs one query for the main entity and N additional queries for related entities.

Solution: **Use JOIN FETCH or batch fetching.**

**Ex.**

@Query("SELECT u FROM User u JOIN FETCH u.orders WHERE u.id = :id")

User findUserWithOrders(@Param("id") Long id);

1. **What is @Transactional, and why is it important?**

@Transactional ensures that a method runs inside a database transaction. If an exception occurs, the transaction is rolled back.

Example:

**@Transactional**

**public void transferMoney(Account from, Account to, double amount) {**

**from.withdraw(amount);**

**to.deposit(amount);**

**}**

1. **What is the difference between persist(), merge(), save(), and update()?**

|  |  |
| --- | --- |
| **Method** | **Purpose** |
| persist(entity) | Adds a new entity to the persistence context but doesn’t immediately insert into the database. |
| merge(entity) | Updates an existing entity or attaches a detached entity to persistence. |
| save(entity) | Immediately inserts into the database (Hibernate-specific). |
| update(entity) | Updates an existing entity (Hibernate-specific). |

1. **How does Hibernate handle optimistic and pessimistic locking?**
2. **Optimistic Locking** – Uses @Version to detect conflicts when multiple transactions update the same entity.

**@Version**

**private int version;**

1. **Pessimistic Locking – Locks the entity to prevent concurrent updates**
   1. **Pessimistic Locking – Locks the entity to prevent concurrent updates**

**@Lock(LockModeType.PESSIMISTIC\_WRITE)**

1. **What is the difference between Criteria API, HQL, and JPQL?**

| **Query Type** | **Description** |
| --- | --- |
| **HQL** | Hibernate-specific query language. |
| **JPQL** | JPA standard query language, independent of Hibernate. |
| **Criteria API** | Programmatic way to build queries using Java objects. |

Example of Criteria API:

CriteriaBuilder cb = entityManager.getCriteriaBuilder();

CriteriaQuery<User> cq = cb.createQuery(User.class);

Root<User> root = cq.from(User.class);

cq.select(root).where(cb.equal(root.get("username"), "john"));

List<User> users = entityManager.createQuery(cq).getResultList();