🔹 A. Hibernate Core Concepts (1–15)

1. **What is Hibernate and how is it different from JDBC?**
    - Hibernate is an ORM (Object Relational Mapping) framework for Java that maps Java objects to relational database tables. It abstracts the complexity of JDBC by eliminating the need for repetitive SQL queries and handling object persistence. Unlike JDBC, Hibernate provides features like caching, lazy loading, and automatic table creation.
2. **Explain the concept of ORM in Hibernate.**
    - ORM is a technique that allows developers to access, manipulate, and manage data between Java objects and a relational database without writing SQL queries. Hibernate uses annotations or XML mappings to define the relationship between Java classes and database tables.
3. **What is the role of the `SessionFactory` and `Session` in Hibernate?**
    - `SessionFactory` is a heavyweight object responsible for creating `Session` instances and is typically created once per application. `Session` is a lightweight object that represents a single-threaded unit of work with the database and is used to perform CRUD operations.
4. **Difference between `get()` and `load()` methods in Hibernate?**
    - `get()` retrieves the actual object and returns `null` if not found. `load()` returns a proxy object and throws `ObjectNotFoundException` if the object is accessed but doesn't exist. `load()` supports lazy loading.
5. **What are transient, persistent, and detached states in Hibernate?**
    - Transient: Object not associated with any Hibernate session or database record.
    - Persistent: Object is associated with a Hibernate session and is tracked for changes.
    - Detached: Object was persistent but the session is closed; changes won’t be auto-synced to the DB.
6. **What is the use of `@Entity`, `@Table`, and `@Id` annotations?**

- `@Entity` marks the class as a persistent entity. `@Table` allows specifying the table name. `@Id` defines the primary key for the entity.

7. **How is the Hibernate cache different from the database cache?**
    - Hibernate cache (1st and 2nd level) works at the application level, caching Java objects, reducing DB hits. DB cache is handled by the DB engine and stores query results or table data in memory.

8. **What are the types of fetching strategies in Hibernate?**
    - Hibernate offers Lazy and Eager fetching. Lazy loads associated data only when needed. Eager fetches all data at once, potentially increasing initial load time.

9. **Explain the difference between lazy and eager fetching.**
    - Lazy fetching loads related data only when it is accessed. Eager fetching retrieves all related data during the initial query, which may lead to unnecessary data loading.

10. **What is cascading in Hibernate? Explain different types.**

- Cascading propagates operations from parent to child entities automatically. Types: ALL, PERSIST, MERGE, REMOVE, DETACH, REFRESH. Useful in saving or deleting associated objects together.

11. **What is the use of `@OneToMany` and `@ManyToOne` mappings?**

- These annotations define the cardinality of relationships between entities. `@OneToMany` indicates one entity is related to many, and `@ManyToOne` is the reverse. Useful for managing bidirectional relationships.

12. **How does Hibernate handle relationships between entities?**

- Using annotations and foreign key mappings (`@JoinColumn`, `@OneToMany`, etc.), Hibernate maps and maintains relationships like one-to-one, one-to-many, and many-to-many.

13. **Explain optimistic and pessimistic locking in Hibernate.**

- Optimistic locking uses versioning to avoid conflicts without DB locks. Pessimistic locking acquires DB locks to prevent concurrent updates, useful in critical operations.

14. **How do you configure Hibernate in a Spring Boot project?**

- By adding dependencies (`spring-boot-starter-data-jpa`, DB drivers), and setting properties like `spring.datasource.*` and `spring.jpa.*` in `application.properties`. Entities are annotated with `@Entity`, and repositories extend `JpaRepository`.

15. **What is the difference between HQL and Criteria API?**

- HQL (Hibernate Query Language) is string-based and similar to SQL. Criteria API is a type-safe, object-oriented way to construct queries dynamically.

🔹 B. Hibernate Transactions & Performance (16–25)

16. **How are transactions managed in Hibernate?**

- Transactions can be managed programmatically using the `Transaction` interface or declaratively using Spring's `@Transactional`. This ensures operations are atomic and can be committed or rolled back.

17. **What is the use of the `@Transactional` annotation?**

- It marks methods as transactional. Spring manages transaction lifecycle, handling commit/rollback automatically based on runtime exceptions.

18. **How do you handle connection pooling in Hibernate?**

- Use built-in pooling or third-party libraries like HikariCP (recommended in Spring Boot), C3P0, or DBCP by configuring them in `application.properties`.

19. **What are first-level and second-level caching?**

- First-level cache is per Hibernate session and mandatory. Second-level cache is optional and shared across sessions. Speeds up access to frequently read data.

20. **How to enable and use second-level cache with EhCache?**

- Add EhCache dependency, configure cache regions in XML or properties, and annotate entities with `@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)`.

21. **What is dirty checking in Hibernate?**

- Hibernate automatically tracks changes made to persistent objects and synchronizes them with the database during `flush()`.

22. **How does Hibernate handle N+1 select problems? How to solve them?**

- Occurs when each parent entity triggers an additional query for its children. Use `JOIN FETCH`, `@BatchSize`, or DTO projections to optimize.

23. **How do you optimize batch inserts/updates in Hibernate?**

- Enable JDBC batching with `hibernate.jdbc.batch_size`, and use manual flushing and clearing of session to manage memory.

24. **What is Hibernate Validator? How do you use it?**

- A framework for bean validation. Add annotations like `@NotNull`, `@Size`, and use `ValidatorFactory` to programmatically validate.

25. **How do you log the actual SQL queries executed by Hibernate?**

- Use `spring.jpa.show-sql=true`, `spring.jpa.properties.hibernate.format_sql=true`, and optionally set logging level for `org.hibernate.SQL` and `org.hibernate.type`.

🔹 C. HQL, JPQL & Criteria (26–35)

26. **Difference between HQL and SQL in Hibernate?**

- HQL is object-oriented, using entity and field names. SQL is database-specific. HQL enables DB-agnostic queries and auto-mapping.

27. **How do you write a join query using HQL?**

- `FROM Order o JOIN o.customer c WHERE c.name = 'John'`

- Uses mapped relationships instead of table joins.

28. **How to use named queries in Hibernate?**

- Declare with `@NamedQuery(name = ..., query = ...)` on entity class. Retrieve using `session.getNamedQuery("name")`.

29. **Explain the Criteria API with an example.**

- Provides a programmatic way to build queries:

```
CriteriaBuilder cb = em.getCriteriaBuilder();
CriteriaQuery<Employee> cq = cb.createQuery(Employee.class);
Root<Employee> root = cq.from(Employee.class);
cq.select(root);
List<Employee> result = em.createQuery(cq).getResultList();
```

30. **How to use projections in Criteria queries?**

- Use `cb.select()` with specific fields or DTO constructors:

```
cq.select(cb.construct(EmployeeDTO.class, root.get("name"), root.get("salary")));
```

31. **How can you perform pagination with HQL?**

- Use `query.setFirstResult(offset)` and `query.setMaxResults(limit)`.

32. **What is the purpose of `fetch join` in HQL?**

- Eagerly loads associations to avoid N+1 problems:

```
SELECT e FROM Department d JOIN FETCH d.employees e
```

33. **How do you perform update/delete using HQL?**

- HQL allows DML operations:

```
session.createQuery("UPDATE Employee e SET e.salary = :salary WHERE e.id =
:id").setParameter("salary", 50000).executeUpdate();
```

34. **Can we call native SQL in Hibernate? How?**

- Yes, using `createNativeQuery("SELECT * FROM employee")`, useful when performance or DB-specific features are needed.

35. **How do you handle result mapping in native queries?**

- Map using `@SqlResultSetMapping`, DTOs, or `addEntity()` in query:

```
session.createNativeQuery(sql).addEntity(Employee.class);
```

🔹 D. Database Querying (SQL & Advanced Use Cases) (36–50)

36. **How do you find the second highest salary in a table?**

- SQL: `SELECT MAX(salary) FROM employees WHERE salary < (SELECT MAX(salary) FROM employees);`

37. **What are `INNER JOIN`, `LEFT JOIN`, `RIGHT JOIN`, and `FULL JOIN` with examples?**

- INNER: Returns matching records.
- LEFT: All from left + matched right.
- RIGHT: All from right + matched left.
- FULL: All records from both sides.

38. **What is the difference between `WHERE` and `HAVING`?**

- `WHERE` filters before grouping. `HAVING` filters after aggregation.

39. **How to optimize a slow SQL query?**

- Add indexes, use EXPLAIN plans, optimize joins, limit selected fields, avoid subqueries.

40. **What is an index? How does it impact query performance?**

- Indexes speed up read operations by allowing faster lookup. However, they add overhead during write operations.

41. **What are window functions? Give an example.**

- Perform calculations across a set of table rows:

```
SELECT name, salary, RANK() OVER (ORDER BY salary DESC) FROM employees;
```

42. **What is the difference between `EXISTS`, `IN`, and `JOIN`?**

- `EXISTS`: True if subquery returns rows. Efficient for correlated subqueries.
- `IN`: Checks if a value is in a list. Can be slower for large lists.
- `JOIN`: Combines rows across tables based on a condition.

43. **How do you prevent SQL injection in Hibernate?**

- Always use parameterized queries with `setParameter()` and avoid string concatenation in queries.

44. **Explain ACID properties in the context of database transactions.**

- Atomicity: All or nothing.
- Consistency: Valid state transitions.
- Isolation: Concurrent transactions don't interfere.
- Durability: Committed changes are permanent.

45. **What is a CTE (Common Table Expression)? Use case?**

- Temporary named result set used to simplify complex queries or enable recursion:

```
WITH dept_avg AS (SELECT dept_id, AVG(salary) FROM employees GROUP BY dept_id)
```

46. **How to write recursive queries in SQL?**

- Use CTEs with recursion:

WITH RECURSIVE employee_hierarchy AS (...) SELECT * FROM employee_hierarchy;

47. **What is the difference between `ROWNUM`, `LIMIT`, and `TOP`?**

- `ROWNUM`: Oracle.
- `LIMIT`: MySQL, PostgreSQL.
- `TOP`: SQL Server. All used to limit result sets.

48. **How do you write a query to find duplicate records in a table?**

SELECT name, COUNT(*) FROM users GROUP BY name HAVING COUNT(*) > 1;

49. **What are aggregate functions in SQL? How are they used in Hibernate?**

- Functions: `SUM()`, `AVG()`, `MIN()`, `MAX()`, `COUNT()`.
- Hibernate HQL: `SELECT COUNT(e) FROM Employee e`

50. **Explain normalization and its types.**

- Reduces redundancy and ensures data integrity.
- 1NF: Atomic columns.
- 2NF: No partial dependencies.
- 3NF: No transitive dependencies.
- BCNF: Every determinant is a candidate key.