**Spring Data JPA HandsOn-3**

**1. Demonstrate writing Hibernate Query Language and Native Query**

**HQL stands for Hibernate Query Language, JPQL stands for Java Persistence Query Language, Compare HQL and JPQL, @Query annotation, HQL fetch keyword, aggregate functions in HQL, Native Query, nativeQuery attribute**

What is HQL?

HQL (Hibernate Query Language) is similar to SQL but operates on Java objects (entities) and not database tables.  
  
What is JPQL?

JPQL (Java Persistence Query Language) is the JPA standard query language, almost identical to HQL.

Hibernate uses HQL,  
Spring Data JPA uses JPQL (by default)  
  
**Entity Example**

import jakarta.persistence.\*;  
@Entity

public class Product {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;  
 private String name;

private double price;  
 private String category;

}  
**Repository Using HQL / JPQL Queries**

import org.springframework.data.jpa.repository.\*;

import org.springframework.data.repository.query.Param;

import java.util.List;

public interface ProductRepository extends JpaRepository<Product, Long> {

// JPQL / HQL query

@Query("SELECT p FROM Product p WHERE p.category = :category")

List<Product> findByCategoryJPQL(@Param("category") String category);

// HQL with aggregate function  
 @Query("SELECT COUNT(p) FROM Product p WHERE p.category = :category")

long countByCategory(@Param("category") String category);

// HQL with fetch keyword (Eager fetch relationships)

@Query("SELECT p FROM Product p JOIN FETCH p.category") // if category were an entity

List<Product> fetchProductsWithCategory();

// Native SQL query

@Query(value = "SELECT \* FROM product WHERE price > :price", nativeQuery = true)

List<Product> findProductsNative(@Param("price") double price);

}  
**Example Usage in a Spring Boot App**

@SpringBootApplication

public class App implements CommandLineRunner {

@Autowired

private ProductRepository repo;

public static void main(String[] args) {

SpringApplication.run(App.class, args);

}

@Override

public void run(String... args) {

repo.save(new Product(null, "TV", 30000, "Electronics"));

repo.save(new Product(null, "Shoes", 2000, "Clothing"));

repo.save(new Product(null, "Laptop", 80000, "Electronics"));

System.out.println("By Category (JPQL):");

repo.findByCategoryJPQL("Electronics").forEach(p -> System.out.println(p.getName()));

System.out.println("Native SQL:");

repo.findProductsNative(5000).forEach(p -> System.out.println(p.getName()));

long count = repo.countByCategory("Electronics");

System.out.println("Count by category: " + count);

}

}  
**2. Explain the need and benefit of Criteria Query**

**Scenarios where Criteria Query helps, CriteriaBuilder, Criteria Query, Root, TypedQuery  
  
Criteria Query**

A type-safe, object-oriented, and dynamic way to create queries in JPA using Java code instead of JPQL strings.

**Scenarios Where Criteria Query Helps**

1. **Dynamic search filters**  
   Example: Search products by optional fields like name, category, price, etc.
2. **Type-safety is important**  
   You want compile-time checking of fields.
3. **Reusable query logic**  
   Build parts of a query and reuse them (e.g., pagination, sorting).  
     
   **Benefits of Criteria Query**

Type-safe  
 Dynamic Querying  
 Reusable  
 No JPQL String  
 Object-Oriented  
 **CriteriaBuilder**

A factory class used to construct criteria queries, expressions, predicates, and ordering in the Criteria API.  
**Root**Represents the main entity/table in the FROM clause of a criteria query.

**TypedQuery**

A query object that is strongly typed with the expected result type; used to execute the criteria query.