1.Demonstrate writing Hibernate Query Language and Native Query

1. **HQL**

- HQL stands for Hibernate Query Language.
- It is an object-oriented query language, similar to SQL but designed for Hibernate.
- HQL works with entities, attributes, and relationships rather than database tables.

2. **JPQL**

- JPQL stands for Java Persistence Query Language.
- It is the standard query language defined by JPA.
- Like HQL, JPQL also works with Java entity objects, not tables.

Using @Query Annotation

- The @Query annotation in Spring Data JPA allows defining custom queries.
- You can write HQL/JPQL or native SQL queries using this annotation.
- Syntax: @Query("SELECT s FROM Student s WHERE s.name = :name")

  List<Student> findByName(@Param("name") String name);

HQL fetch Keyword

- The fetch keyword in HQL is used with JOIN FETCH to perform eager loading.
- It retrieves related entities in a single query.

Aggregate Functions in HQL

- Common aggregate functions supported in HQL:
- COUNT().
- SUM()
- AVG()
- MIN()
- MAX()

Native Query

- Native queries are raw SQL queries written directly for the database.
- Spring Data JPA allows native SQL using @Query with nativeQuery = true.

nativeQuery **Attribute**

- The nativeQuery attribute is used to indicate whether the query is a native SQL query.
- If nativeQuery = true, Spring will treat the query as SQL instead of JPQL.

Example – Repository

```
@Repository
public interface StudentRepository extends JpaRepository<Student, Long> {
    // HQL / JPQL
    @Query("SELECT s FROM Student s WHERE s.department = :dept")
```

```
    List<Student> findByDepartment(@Param("dept") String dept);

    // HQL with fetch keyword

    @Query("SELECT s FROM Student s JOIN FETCH s.department")

    List<Student> fetchStudentsWithDepartment();

    // HQL with aggregate function

    @Query("SELECT COUNT(s) FROM Student s WHERE s.department = :dept")

    Long countByDepartment(@Param("dept") String dept);

    // Native query

    @Query(value = "SELECT * FROM student WHERE age > :age", nativeQuery = true)

    List<Student> findNativeByAgeGreaterThan(@Param("age") int age);

}
```