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

**Spring Data JPA with Spring Boot, Hibernate**

Vaishnavi

6/7/2025

In the world of ORM (Object-Relational Mapping), instead of writing raw SQL to interact with databases, we use **object-oriented query languages** like:

**HQL (Hibernate Query Language)** — Hibernate's query language that works on entity objects, not tables.

**JPQL (Java Persistence Query Language)** — Standardized version of HQL as part of the JPA specification.

Both use **entity classes and field names** instead of table and column names. They support SELECT, UPDATE, DELETE, and **aggregate functions** like AVG(), SUM(), COUNT(). HQL additionally supports INSERT.

Also, **native queries** (raw SQL) can still be used when fine-tuned control is required.

**Objective:**

Understand and compare HQL and JPQL.

Write HQL queries using @Query in Spring Data JPA.

Fetch associated entities using JOIN and FETCH.

Use aggregate functions like AVG().

Write native SQL queries using nativeQuery = true.

Use **Criteria API** to build dynamic queries based on user filters.

Analyze the number and performance of generated SQL queries.

**Implementation:**

### Introduction to HQL and JPQL

HQL is Hibernate's proprietary query language.

JPQL is part of the JPA spec (a subset of HQL).

Use entity names and attributes in queries.

### ****Get All Permanent Employee (HQL)****

**EmployeeRespository.java**

@Query("SELECT e FROM Employee e LEFT JOIN FETCH e.department d LEFT JOIN FETCH e.skillList WHERE e.permanent = true")

List<Employee> getAllPermanentEmployees();

**EmployeeService.java**

public List<Employee> getAllPermanentEmployees() {

return employeeRepository.getAllPermanentEmployees();

}

**MainApplication.java**

public static void testGetAllPermanentEmployees() {

List<Employee>employees=employeeService.getAllPermanentEmployees();

employees.forEach(e -> {

LOGGER.info("Employee: {}", e);

LOGGER.info("Skills: {}", e.getSkillList());

});

}

### Fetch Quiz Attempt Details (HQL with JOIN FETCH)

**AttemptRespository.java**

@Query("SELECT a FROM Attempt a " +

"JOIN FETCH a.user u " +

"JOIN FETCH a.attemptQuestions aq " +

"JOIN FETCH aq.question q " +

"JOIN FETCH aq.attemptOptions ao " +

"JOIN FETCH ao.option o " +

"WHERE u.id = :userId AND a.id = :attemptId")

Attempt getAttemptDetails(@Param("userId") int userId, @Param("attemptId") int attemptId);

**Get Average Salary Using HQL**

**EmployeeRepository.java**

@Query("SELECT AVG(e.salary) FROM Employee e WHERE e.department.id = :id")

double getAverageSalary(@Param("id") int id);

**EmployeeService.java**

public double getAverageSalary(int deptId) {

return employeeRepository.getAverageSalary(deptId);

}

**Main.java**

public static void testGetAverageSalary() {

double avg = employeeService.getAverageSalary(1);

LOGGER.info("Average Salary: {}", avg);

}

**Get All Employees Using Native Query**

**EmployeeRepository.java**

@Query(value = "SELECT \* FROM employee", nativeQuery = true)

List<Employee> getAllEmployeesNative();

**EmployeeService.java**

public List<Employee> getAllEmployeesNative() {

return employeeRepository.getAllEmployeesNative();

}

**Main.java**

public static void testGetAllEmployeesNative() {

List<Employee> empList = employeeService.getAllEmployeesNative();

empList.forEach(e -> LOGGER.info("Employee: {}", e));

}

**Dynamic Search Using Criteria Query**

public List<Product> searchProducts(String os, Double minPrice, Integer minRam) {

CriteriaBuilder cb = entityManager.getCriteriaBuilder();

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

Root<Product> product = cq.from(Product.class);

List<Predicate> predicates = new ArrayList<>();

if (os != null)

predicates.add(cb.equal(product.get("os"), os));

if (minPrice != null)

predicates.add(cb.ge(product.get("price"), minPrice));

if (minRam != null)

predicates.add(cb.ge(product.get("ram"), minRam));

cq.where(predicates.toArray(new Predicate[0]));

return entityManager.createQuery(cq).getResultList();

}