**Spring Data JPA: HQL, Native Queries & Criteria Queries**

**Objectives**

**Demonstrate Writing Hibernate Query Language and Native Query**

* **HQL (Hibernate Query Language)** and **JPQL (Java Persistence Query Language)**
* Compare HQL and JPQL
* @Query annotation
* fetch keyword in HQL
* Aggregate functions in HQL
* Native Query and nativeQuery attribute

**Reference:**

* Hibernate Developer Guide (Chapter 11)
* Features of JPA Queries (Baeldung)

**Explain the Need and Benefit of Criteria Query**

* Scenarios where Criteria Query is helpful
* Components: CriteriaBuilder, CriteriaQuery, Root, TypedQuery

**Reference:**

* Oracle Java EE 6 Tutorial (JPA Criteria API)

**Hands-on 1: Introduction to HQL and JPQL**

* HQL: Hibernate Query Language
* JPQL: Java Persistence Query Language
* Both are object-focused query languages similar to SQL
* JPQL is a subset of HQL
* All JPQL queries are valid HQL queries, but the reverse is not always true
* Both support SELECT, UPDATE, and DELETE
* HQL also supports INSERT

**Reference: Hibernate ORM Dev Guide (Chapter 11)**

**Hands-on 2: Get All Permanent Employees Using HQL**

**Steps:**

1. **HQL Query**

**java**

@Query(value = "SELECT e FROM Employee e WHERE e.permanent = 1")

List<Employee> getAllPermanentEmployees();

* Note: References Java class fields instead of table columns

1. **Logging Employee & Skill Details**

**java**

public static void testGetAllPermanentEmployees() {

LOGGER.info("Start");

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

LOGGER.debug("Permanent Employees:{}", employees);

employees.forEach(e -> LOGGER.debug("Skills:{}", e.getSkillList()));

LOGGER.info("End");

}

1. **SQL Queries Observed in Logs**

* Multiple joins executed (employee, department, employee\_skill, skill)
* Skill details fetched through eager loading

**Optimization: Remove Eager Fetching**

* Remove FetchType.EAGER from Department.employeeList and Employee.skillList
* Observed issues:
  + Skills not fetched
  + Still multiple queries executed

**Further Optimization: Using fetch**

**java**

@Query("SELECT e FROM Employee e left join fetch e.department d left join fetch e.skillList WHERE e.permanent = 1")

* All required data fetched in a **single query**

**Takeaway:**

* join links tables but doesn’t populate Java beans
* fetch is required to load related entities eagerly

**Hands-on 3: Fetch Quiz Attempt Details Using HQL**

**Requirements:**

* Show quiz attempt by a user:
  + Username
  + Attempted Date
  + List of Questions and Options
  + Correct Answers
  + Score for Correct Answers

**Steps:**

1. **Setup Schema**
   * Use MySQL Workbench (quiz.mwb)
   * Forward Engineer to generate and run SQL in ormlearn schema
2. **Define Entities**
   * User, Question, Option, Attempt, AttemptQuestion, AttemptOption
   * Define relationships as per schema
3. **Repositories and Service**
   * AttemptRepository:

java

CopyEdit

public Attempt getAttempt(int userId, int attemptId);

* + AttemptService:

java

CopyEdit

public Attempt getAttempt(int userId, int attemptId);

1. **OrmLearnApplication.java**
   * Add test method to fetch and display attempt details
2. **Write HQL Query**
   * Join entities: user → attempt → attempt\_question → question → attempt\_option → options
   * Use fetch in one-to-many and many-to-many joins

**Sample Output Format:**

pgsql

CopyEdit

What is the extension of the hyper text markup language file?

1) .xhtm 0.0 false

2) .ht 0.0 false

3) .html 1.0 true

4) .htmx 0.0 false

What is the maximum level of heading tag can be used in a HTML page?

1) 5 0.0 false

2) 3 0.0 true

3) 4 0.0 false

4) 6 1.0 false

**Hands-on 4: Get Average Salary Using HQL**

**Basic Average Salary**

**java**

@Query("SELECT AVG(e.salary) FROM Employee e")

double getAverageSalary();

**Filtered by Department**

**java**

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

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

**Key Notes:**

* Use : for query parameters
* Use @Param to bind method parameters
* HQL supports all aggregate functions like AVG, COUNT, MAX, MIN, SUM

**Hands-on 5: Get All Employees Using Native Query**

**Definition**

**java**

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

List<Employee> getAllEmployeesNative();

**Notes:**

* Native Queries are actual SQL queries
* Use them only when HQL is insufficient
* Less portable across different databases

**Hands-on 6: Criteria Query**

**Real-World Scenario:**

* E-commerce search with filters (e.g., Amazon filters on laptops)
* Filters include:
  + Customer review
  + Hard Disk Size
  + RAM Size
  + CPU Speed
  + Operating System
  + Weight
  + CPU

**Challenge:**

* HQL query's WHERE clause varies dynamically based on user filters

**Solution: Criteria Query**

* Allows dynamic construction of queries
* Add filters conditionally using Java code

**Components:**

* CriteriaBuilder
* CriteriaQuery<T>
* Root<T>
* TypedQuery<T>

**Reference:**

* HowToDoInJava: Hibernate Criteria Queries Tutorial