# SPRING\_DATA\_JPA

Hands on 1-Introduction to HQL and JPQL

HQL

 HQL = Hibernate Query Language

 A powerful, object-oriented query language developed for Hibernate ORM

 Works with entity objects and their property names, *not*

table or column names

 HQL supports SELECT, UPDATE, DELETE, and also

INSERT (which JPQL doesn't)

JPQL

 JPQL = Java Persistence Query Language

 Defined in the JPA (Java Persistence API) speciflcation

 JPQL is a subset of HQL and works similarly

 It supports SELECT, UPDATE, DELETE (but not INSERT)

Sample HQL/JPQL Syntax

// SELECT

Query query = session.createQuery("FROM Employee WHERE salary > :salary");

query.setParameter("salary", 50000); List<Employee> list = query.list();

// UPDATE

Query updateQuery = session.createQuery("UPDATE Employee SET salary = :newSal WHERE id = :empId");

updateQuery.setParameter("newSal", 60000);

updateQuery.setParameter("empId", 1); int result = updateQuery.executeUpdate();

// DELETE

Query deleteQuery = session.createQuery("DELETE FROM Employee WHERE id = :empId");

deleteQuery.setParameter("empId", 1); deleteQuery.executeUpdate();

Hands on 2-Get all permanent employees using HQL

Employee.java

package com.cognizant.ormlearn.model;

import javax.persistence.\*; import java.util.Date; import java.util.HashSet; import java.util.Set;

@Entity

@Table(name = "employee") public class Employee {

@Id

@Column(name = "em\_id") private int id;

@Column(name = "em\_name")

private String name;

@Column(name = "em\_salary") private double salary;

@Column(name = "em\_permanent") private boolean permanent;

@Column(name = "em\_date\_of\_birth") @Temporal(TemporalType.DATE) private Date dateOfBirth;

@ManyToOne @JoinColumn(name = "em\_dp\_id") private Department department;

@ManyToMany

@JoinTable(name = "employee\_skill",

joinColumns = @JoinColumn(name = "es\_em\_id"),

inverseJoinColumns = @JoinColumn(name = "es\_sk\_id"))

private Set<Skill> skillList = new HashSet<>();

// Getters and Setters public int getId() {

return id;

}

public void setId(int id) { this.id = id;

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public double getSalary() { return salary;

}

public void setSalary(double salary) { this.salary = salary;

}

public boolean isPermanent() { return permanent;

}

public void setPermanent(boolean permanent) { this.permanent = permanent;

}

public Date getDateOfBirth() { return dateOfBirth;

}

public void setDateOfBirth(Date dateOfBirth) { this.dateOfBirth = dateOfBirth;

}

public Department getDepartment() { return department;

}

public void setDepartment(Department department) {

this.department = department;

}

public Set<Skill> getSkillList() { return skillList;

}

public void setSkillList(Set<Skill> skillList) { this.skillList = skillList;

}

@Override

public String toString() {

return "Employee [id=" + id + ", name=" + name + ", salary=" + salary + ", permanent=" + permanent +

", dateOfBirth=" + dateOfBirth + ", department=" + (department != null ? department.getName() : "null") + "]";

}

}

Department.java

package com.cognizant.ormlearn.model;

import javax.persistence.\*; import java.util.Set;

@Entity

@Table(name = "department") public class Department {

@Id

@Column(name = "dp\_id") private int id;

@Column(name = "dp\_name") private String name;

@OneToMany(mappedBy = "department") private Set<Employee> employeeList;

// Getters and Setters

public int getId() { return id;

}

public void setId(int id) { this.id = id;

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public Set<Employee> getEmployeeList() { return employeeList;

}

public void setEmployeeList(Set<Employee> employeeList) {

this.employeeList = employeeList;

}

}

Skill.java

package com.cognizant.ormlearn.model;

import javax.persistence.\*; import java.util.Set;

@Entity @Table(name = "skill") public class Skill {

@Id

@Column(name = "sk\_id") private int id;

@Column(name = "sk\_name") private String name;

@ManyToMany(mappedBy = "skillList") private Set<Employee> employeeList;

// Getters and Setters

public int getId() { return id;

}

public void setId(int id) { this.id = id;

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public Set<Employee> getEmployeeList() { return employeeList;

}

public void setEmployeeList(Set<Employee> employeeList) {

this.employeeList = employeeList;

}

@Override

public String toString() {

return "Skill [id=" + id + ", name=" + name + "]";

}

}

EmployeeRepository.java

package com.cognizant.ormlearn.repository;

import com.cognizant.ormlearn.model.Employee;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Query; import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

@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

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Employee; import java.util.List;

public interface EmployeeService { List<Employee> getAllPermanentEmployees();

}

EmployeeServiceImpl.java

package com.cognizant.ormlearn.service;

import com.cognizant.ormlearn.model.Employee; import

com.cognizant.ormlearn.repository.EmployeeRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service; import java.util.List;

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public List<Employee> getAllPermanentEmployees() { return

employeeRepository.getAllPermanentEmployees();

}

}

OrmLearnApplication.java

package com.cognizant.ormlearn;

import com.cognizant.ormlearn.model.Employee;

import com.cognizant.ormlearn.service.EmployeeService; import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplic ation;

import org.springframework.context.ApplicationContext; import java.util.List;

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

private static EmployeeService employeeService;

public static void main(String[] args) { ApplicationContext context =

SpringApplication.run(OrmLearnApplication.class, args);

employeeService = context.getBean(EmployeeService.class);

testGetAllPermanentEmployees(); // Run the test method

}

public static void testGetAllPermanentEmployees() { LOGGER.info("Start");

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

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

for (Employee e : employees) {

LOGGER.debug("Employee: {}", e.getName()); LOGGER.debug("Department: {}",

e.getDepartment().getName());

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

}

LOGGER.info("End");

}

}

Hands on 3-Fetch quiz attempt details using HQL

## AttemptKey.java

@Embeddable

public class AttemptKey implements Serializable { @Column(name = "user\_id") private int userId; @Column(name = "attempt\_id") private int attemptId;

// Constructors, equals(), hashCode(), getters/setters

}

#### AttemptQuestionKey.java

@Embeddable

public class AttemptQuestionKey implements Serializable { @Embedded private AttemptKey attemptKey; @Column(name = "question\_id") private int questionId;

// Constructors, equals(), hashCode(), getters/setters

}

AttemptOptionKey.java

@Embeddable

public class AttemptOptionKey implements Serializable {

@Embedded private AttemptQuestionKey attemptQuestionKey; @Column(name = "option\_id") private int optionId;

// Constructors, equals(), hashCode(), getters/setters

}

User.java

@Entity

@Table(name = "user") public class User {

@Id @Column(name = "user\_id") private int id; @Column(name = "username") private String username;

@OneToMany(mappedBy = "user") private Set<Attempt> attempts = new HashSet<>();

// getters/setters

}

Attempt.java

@Entity

@Table(name = "attempt") public class Attempt {

@EmbeddedId private AttemptKey id;

@ManyToOne @MapsId("userId") @JoinColumn(name = "user\_id") private User user; @Column(name = "attempted\_date") private LocalDateTime attemptedDate;

@OneToMany(mappedBy = "attempt") private Set<AttemptQuestion> questions = new HashSet<>();

// getters/setters

}

## AttemptQuestion.java

@Entity

@Table(name = "attempt\_question") public class AttemptQuestion {

@EmbeddedId private AttemptQuestionKey id;

@ManyToOne @MapsId("attemptKey") @JoinColumns({ @JoinColumn(name = "user\_id"), @JoinColumn(name = "attempt\_id")

}) private Attempt attempt;

@ManyToOne @MapsId("questionId") @JoinColumn(name = "question\_id") private Question question; @OneToMany(mappedBy = "attemptQuestion") private Set<AttemptOption> selectedOptions = new

HashSet<>();

// getters/setters

}

AttemptOption.java

@Entity

@Table(name = "attempt\_option") public class AttemptOption {

@EmbeddedId private AttemptOptionKey id;

@ManyToOne @MapsId("attemptQuestionKey") @JoinColumns({

@JoinColumn(name = "user\_id"), @JoinColumn(name = "attempt\_id"), @JoinColumn(name = "question\_id")

}) private AttemptQuestion attemptQuestion;

@ManyToOne @MapsId("optionId") @JoinColumn(name = "option\_id") private QuizOption option; @Column(name = "is\_selected") private boolean selected;

// getters/setters

}

#### Question.java

@Entity

@Table(name = "question") public class Question {

@Id @Column(name = "question\_id") private int id;

@Column(name = "text") private String text; @Column(name = "score") private double score;

@OneToMany(mappedBy = "question") private Set<QuizOption> options = new HashSet<>();

// getters/setters

}

QuizOption.java

@Entity

@Table(name = "options") public class QuizOption {

@Id @Column(name = "option\_id") private int id; @Column(name = "option\_text") private String text; @Column(name = "is\_correct") private boolean correct;

@ManyToOne @JoinColumn(name = "question\_id") private Question question;

// getters/setters

}

## OrmLearnApplication.java

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired private AttemptService attemptService;

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) {

Attempt a = attemptService.getAttempt(1, 1);

LOGGER.info("Username: {}", a.getUser().getUsername()); LOGGER.info("Attempted Date: {}", a.getAttemptedDate()); for (AttemptQuestion aq : a.getQuestions()) {

Question q = aq.getQuestion(); LOGGER.info("\n{}", q.getText()); int idx = 1;

for (QuizOption opt : q.getOptions()) {

boolean isSelected = aq.getSelectedOptions()

.stream().anyMatch(ao -> ao.getOption().getId() == opt.getId() && ao.isSelected()); LOGGER.info(" {}) {} {} {}", idx++, opt.getText(), q.getScore(), isSelected);

}

}

}

}

Hands on 4-Get average salary using HQL

## Department.java

@Entity

@Table(name = "department") public class Department {

@Id

@Column(name = "id") private int id;

@Column(name = "name") private String name;

@OneToMany(mappedBy = "department") private List<Employee> employees;

// Getters and Setters

}

Employee.java

@Entity

@Table(name = "employee") public class Employee {

@Id

@Column(name = "id") private int id;

@Column(name = "name") private String name;

@Column(name = "salary") private double salary;

@ManyToOne

@JoinColumn(name = "department\_id") private Department department;

// Getters and Setters

}

EmployeeRepository.java

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

// Average salary across all departments

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

// Average salary for a specific department

@Query("SELECT AVG(e.salary) FROM Employee e WHERE e.department.id = :id") double getAverageSalary(@Param("id") int departmentId);

}

## EmployeeService.java

public interface EmployeeService {

double getAverageSalary();

double getAverageSalaryByDepartment(int departmentId);

}

EmployeeServiceImpl.java

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public double getAverageSalary() {

return employeeRepository.getAverageSalary();

}

@Override

public double getAverageSalaryByDepartment(int departmentId) { return employeeRepository.getAverageSalary(departmentId);

}

}

OrmLearnApplication.java

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired

private EmployeeService employeeService;

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) { testAverageSalary();

testAverageSalaryByDepartment();

}

private void testAverageSalary() {

double avg = employeeService.getAverageSalary();

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

}

private void testAverageSalaryByDepartment() {

int deptId = 1; // Replace with valid department ID in your DB

double avg = employeeService.getAverageSalaryByDepartment(deptId);

LOGGER.info("Average Salary for Department ID {}: {}", deptId, avg);

}

}

Hands on 5-Get all employees using Native Query

Employee.java

@Entity

@Table(name = "employee") public class Employee {

@Id

@Column(name = "id") private int id;

@Column(name = "name") private String name;

@Column(name = "salary") private double salary;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

// Getters and Setters

}

EmployeeRepository.java

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

@Query(value = "SELECT \* FROM employee", nativeQuery = true) List<Employee> getAllEmployeesNative();

}

EmployeeService.java

public interface EmployeeService { List<Employee> getAllEmployeesNative();

}

### EmployeeServiceImpl.java

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public List<Employee> getAllEmployeesNative() {

return employeeRepository.getAllEmployeesNative();

}

}

OrmLearnApplication.java

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired

private EmployeeService employeeService;

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) throws Exception { testGetAllEmployeesNative();

}

private void testGetAllEmployeesNative() {

LOGGER.info("Start: testGetAllEmployeesNative");

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

for (Employee emp : employeeList) {

LOGGER.info("Employee: {} | Salary: {}", emp.getName(), emp.getSalary());

}

LOGGER.info("End: testGetAllEmployeesNative");

}

}

# output

Start: testGetAllEmployeesNative Employee: John | Salary: 50000.0 Employee: Alice | Salary: 60000.0

...

End: testGetAllEmployeesNative

Hands on 6-Criteria Query

### Product.java

@Entity

@Table(name = "product") public class Product {

@Id

private int id;

private String name;

private String cpu;

private int ramSize;

private String operatingSystem;

private double cpuSpeed;

private double customerReview;

private double weight;

private String hardDiskSize;

// Getters and Setters

}

ProductRepositoryCustom.java

public interface ProductRepositoryCustom {

List<Product> searchProducts(String name, Integer ramSize, String cpu, String os, Double review, Double cpuSpeed);

}

## ProductRepositoryImpl.java

@Repository

public class ProductRepositoryImpl implements ProductRepositoryCustom {

@PersistenceContext

private EntityManager entityManager;

@Override

public List<Product> searchProducts(String name, Integer ramSize, String cpu, String os, Double review, Double cpuSpeed) {

CriteriaBuilder cb = entityManager.getCriteriaBuilder();

CriteriaQuery<Product> query = cb.createQuery(Product.class); Root<Product> root = query.from(Product.class);

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

if (name != null && !name.isEmpty()) {

predicates.add(cb.like(cb.lower(root.get("name")), "%" + name.toLowerCase() + "%"));

}

if (ramSize != null) {

predicates.add(cb.ge(root.get("ramSize"), ramSize));

}

if (cpu != null && !cpu.isEmpty()) {

predicates.add(cb.equal(root.get("cpu"), cpu));

}

if (os != null && !os.isEmpty()) {

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

}

if (review != null) {

predicates.add(cb.ge(root.get("customerReview"), review));

}

if (cpuSpeed != null) {

predicates.add(cb.ge(root.get("cpuSpeed"), cpuSpeed));

}

query.select(root).where(cb.and(predicates.toArray(new Predicate[0]))); return entityManager.createQuery(query).getResultList();

}

}

ProductRepository.java

public interface ProductRepository extends JpaRepository<Product, Integer>, ProductRepositoryCustom {

}

#### ProductService.java

public interface ProductService {

List<Product> dynamicSearch(String name, Integer ramSize, String cpu, String os, Double review, Double cpuSpeed);

}

ProductServiceImpl.java

@Service

public class ProductServiceImpl implements ProductService {

@Autowired

private ProductRepository productRepository;

@Override

public List<Product> dynamicSearch(String name, Integer ramSize, String cpu, String os, Double review, Double cpuSpeed) {

return productRepository.searchProducts(name, ramSize, cpu, os, review, cpuSpeed);

}

}

OrmLearnApplication.java

@Autowired

private ProductService productService;

private void testDynamicSearch() {

List<Product> results = productService.dynamicSearch("laptop", 8, "Intel i5", "Windows", 4.0, null); for (Product product : results) {

System.out.println(product.getName() + " - " + product.getRamSize() + "GB RAM - " + product.getCpu());

}

}