**Question 01:**

Consider a database schema with two entities: **Employee** and **Department**. The **Employee** entity has attributes such as **id** (primary key), **name**, **salary**, and a many-to-one relationship with the **Department** entity. The **Department** entity has attributes **id** (primary key) and **name**. Formulate a set of tasks using the Criteria API to perform the following operations:

1. Retrieve a list of employees whose salary is within a specified range.
2. Obtain a list of employees ordered by their salary in descending order.
3. Retrieve employees based on the name of their associated department.
4. Retrieve employees whose names match a given pattern.

Provide the corresponding answers with well-explained examples in Java using the Hibernate Criteria API, ensuring proper session management and exception handling.

Answer:

import javax.persistence.\*;

import java.util.List;

@Entity

@Table(name = "departments")

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private Long id;

@Column(name = "name")

private String name;

@OneToMany(mappedBy = "department")

private List<Employee> employees;

// Constructors, getters, and setters

}

import javax.persistence.\*;

@Entity

@Table(name = "employees")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private Long id;

@Column(name = "name")

private String name;

@Column(name = "salary")

private double salary;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

// Constructors, getters, and setters

}

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import javax.persistence.criteria.\*;

import java.util.List;

public class CriteriaApiExample {

private static SessionFactory sessionFactory;

static {

try {

// Initialize Hibernate SessionFactory

Configuration configuration = new Configuration().configure();

sessionFactory = configuration.buildSessionFactory();

} catch (Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

try {

// First add some rows

// Task 1: Retrieve employees with a specified salary range

List<Employee> employeesInSalaryRange = getEmployeesInSalaryRange(50000, 80000);

System.out.println("Employees in salary range:");

printEmployees(employeesInSalaryRange);

// Task 2: Retrieve employees ordered by salary in descending order

List<Employee> employeesOrderBySalaryDesc = getEmployeesOrderBySalaryDesc();

System.out.println("\nEmployees ordered by salary in descending order:");

printEmployees(employeesOrderBySalaryDesc);

// Task 3: Retrieve employees based on the name of their associated department

List<Employee> employeesByDepartment = getEmployeesByDepartmentName("IT");

System.out.println("\nEmployees in the IT department:");

printEmployees(employeesByDepartment);

// Task 4: Retrieve employees whose names match a given pattern

List<Employee> employeesByNamePattern = getEmployeesByNamePattern("John");

System.out.println("\nEmployees with names matching the pattern 'John':");

printEmployees(employeesByNamePattern);

} finally {

// Close the sessionFactory to release resources

if (sessionFactory != null) {

sessionFactory.close();

}

}

}

public static List<Employee> getEmployeesInSalaryRange(double minSalary, double maxSalary) {

}

public static List<Employee> getEmployeesOrderBySalaryDesc() {

}

public static List<Employee> getEmployeesByDepartmentName(String departmentName) {

}

public static List<Employee> getEmployeesByNamePattern(String namePattern) {

}

private static void printEmployees(List<Employee> employees) {

}

}

**Question 02:**

Consider a database schema with two entities: **Course** and **Student**. The **Course** entity has attributes such as **id** (primary key) and **name**, and a one-to-many relationship with the **Student** entity. The **Student** entity has attributes **id** (primary key), **name**, **grade**, and a many-to-one relationship with the **Course** entity.

**Answer:**

import javax.persistence.\*;

import java.util.List;

@Entity

@Table(name = "courses")

public class Course {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private Long id;

@Column(name = "name")

private String name;

@OneToMany(mappedBy = "course")

private List<Student> students;

// Constructors, getters, and setters

}

import javax.persistence.\*;

@Entity

@Table(name = "students")

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private Long id;

@Column(name = "name")

private String name;

@Column(name = "grade")

private int grade;

@ManyToOne

@JoinColumn(name = "course\_id")

private Course course;

// Constructors, getters, and setters

}

**To add some rows**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.util.Date;

public class AddRowsExample {

private static SessionFactory sessionFactory;

static {

try {

// Initialize Hibernate SessionFactory

Configuration configuration = new Configuration().configure();

sessionFactory = configuration.buildSessionFactory();

} catch (Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

try {

// Open a session

Session session = sessionFactory.openSession();

Transaction transaction = session.beginTransaction();

// Add 100 courses

for (int i = 1; i <= 100; i++) {

Course course = new Course();

course.setName("Course " + i);

session.save(course);

}

// Add 500 students

for (int i = 1; i <= 500; i++) {

Student student = new Student();

student.setName("Student " + i);

student.setEnrollmentDate(getRandomEnrollmentDate());

// Assign a random course to the student

Course randomCourse = getRandomCourse(session);

student.setCourse(randomCourse);

session.save(student);

}

// Commit the transaction

transaction.commit();

System.out.println("Rows added successfully!");

} catch (Exception e) {

e.printStackTrace();

} finally {

// Close the sessionFactory to release resources

if (sessionFactory != null) {

sessionFactory.close();

}

}

}

private static Date getRandomEnrollmentDate() {

long startDate = toDate("2020-01-01").getTime();

long endDate = System.currentTimeMillis();

long randomDate = startDate + (long) (Math.random() \* (endDate - startDate));

return new Date(randomDate);

}

private static Date toDate(String dateString) {

try {

return new SimpleDateFormat("yyyy-MM-dd").parse(dateString);

} catch (ParseException e) {

e.printStackTrace();

return null;

}

}

private static Course getRandomCourse(Session session) {

// Retrieve a random course from the database

long courseId = (long) (Math.random() \* 100) + 1;

return session.get(Course.class, courseId);

}

}

Formulate a task using the Hibernate Criteria API to perform the following operation:

1. Retrieve a list of students who are enrolled in a specific course and have a grade greater than or equal to a specified minimum value.
2. Retrieve a list of students who are enrolled in courses with names starting with a specified prefix and joined the course before a certain date.
3. Retrieve a list of courses with the highest number of enrolled students.
4. Retrieve a list of courses with the highest number of enrolled students.
5. Retrieve a list of courses that have at least one student enrolled before a specified date and also have at least one student enrolled after a different specified date.
6. Retrieve a list of students who are enrolled in courses that have the letter 'A' in their names.
7. Retrieve a list of courses that have an average student grade greater than a specified value.
8. Retrieve a list of students who are enrolled in a specific course and have a grade greater than or equal to a specified minimum value.
9. Retrieve a list of courses that have students with grades in a specified range (e.g., between 70 and 90).
10. Retrieve a list of students who are not enrolled in any course.

**Question 03:**

Suppose you have a database schema with two entities: **Book** and **Author**. The **Book** entity has attributes such as **id** (primary key), **title**, **publicationYear**, and a many-to-one relationship with the **Author** entity. The **Author** entity has attributes **id** (primary key), **name**, and a one-to-many relationship with the **Book** entity.

**Answer:**

**Book.java**

import javax.persistence.\*;

import java.util.Objects;

@Entity

@Table(name = "books")

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private Long id;

@Column(name = "title")

private String title;

@Column(name = "publication\_year")

private int publicationYear;

@ManyToOne

@JoinColumn(name = "author\_id")

private Author author;

// Constructors, getters, and setters

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Book book = (Book) o;

return Objects.equals(id, book.id);

}

@Override

public int hashCode() {

return Objects.hash(id);

}

}

**Author.java**

import javax.persistence.\*;

import java.util.List;

import java.util.Objects;

@Entity

@Table(name = "authors")

public class Author {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

@Column(name = "id")

private Long id;

@Column(name = "name")

private String name;

@OneToMany(mappedBy = "author")

private List<Book> books;

// Constructors, getters, and setters

@Override

public boolean equals(Object o) {

if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false;

Author author = (Author) o;

return Objects.equals(id, author.id);

}

@Override

public int hashCode() {

return Objects.hash(id);

}

}

**To insert rows into books and authors**

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import java.util.Arrays;

import java.util.List;

import java.util.Random;

public class AddRealRowsExample {

private static SessionFactory sessionFactory;

static {

try {

// Initialize Hibernate SessionFactory

Configuration configuration = new Configuration().configure();

sessionFactory = configuration.buildSessionFactory();

} catch (Exception e) {

e.printStackTrace();

}

}

public static void main(String[] args) {

try {

// Open a session

Session session = sessionFactory.openSession();

Transaction transaction = session.beginTransaction();

// Add real authors and books data (using fictional names)

List<String> authorNames = Arrays.asList(

"Alice Johnson", "Bob Smith", "Catherine Davis", "David Miller", "Emma Wilson",

"Frank White", "Grace Brown", "Henry Jones", "Ivy Robinson", "Jack Thompson",

"Katherine Lee", "Leo Martinez", "Mia Garcia", "Nathan Hall", "Olivia Wright"

);

List<String> bookTitles = Arrays.asList(

"The Enchanted Forest", "Echoes of Eternity", "Lost in Dreams", "Beyond the Stars",

"The Midnight Serenade", "Whispers in the Wind", "Eternal Love", "The Forgotten Realm",

"A Tale of Two Cities", "The Secret Garden", "Shadows of Destiny", "A Symphony of Stars",

"Ripples of Time", "The Art of Silence", "The Whispering Pines"

);

for (String authorName : authorNames) {

Author author = new Author();

author.setName(authorName);

session.save(author);

for (String bookTitle : bookTitles) {

Book book = new Book();

book.setTitle(bookTitle);

book.setPublicationYear(2020 + new Random().nextInt(5)); // Random publication year between 2020 and 2024

book.setAuthor(author);

session.save(book);

}

}

// Commit the transaction

transaction.commit();

System.out.println("Rows added successfully!");

} catch (Exception e) {

e.printStackTrace();

} finally {

// Close the sessionFactory to release resources

if (sessionFactory != null) {

sessionFactory.close();

}

}

}

}

Formulate a task using the Hibernate Criteria API to perform the following operation:

1. Retrieve a list of books published after a certain year, written by authors who have more than a specified number of books.
2. Retrieve a list of authors who have written books with titles containing a specified keyword, and the total number of books they've written is greater than a specified threshold.
3. Retrieve a list of books written by authors who have more than a specified number of books, and the average publication year of their books is older than a given year.
4. Retrieve a list of authors who have written books with titles containing a specified keyword, and the average publication year of their books is within a given range.
5. Retrieve a list of books written by authors who have at least one book published before a specified year and at least one book published after another specified year.
6. Retrieve a list of authors who have written books with titles starting with a specified prefix, and the total number of books they've written is within a specified range.
7. Retrieve a list of authors who have written books with titles ending with a specified suffix, and the average publication year of their books is within a given range.
8. Retrieve a list of books written by authors who have written books with titles containing a specified keyword, and the total number of books they've written is less than a specified threshold.
9. Retrieve a list of authors who have written books published within a specified range of years, and the average publication year of their books is greater than a given year.
10. Retrieve a list of authors who have written books with titles containing a specified keyword, and the total number of books they've written is equal to a specified number.
11. Retrieve a list of books published within a specified range of years, and the authors of these books have written books with titles containing a specified keyword.
12. Retrieve a list of authors who have written books with titles containing a specified keyword and were published within a given range of years. Additionally, include in the result the total number of books each author has written, the earliest and latest publication year of their books, and the average publication year of all their books. Sort the result based on the author's name in ascending order.
13. Retrieve a list of authors who have written books in a specified genre, and the books were published within a given range of years. Additionally, include in the result the total number of books each author has written, the total number of books in the specified genre, and the average publication year of their books in that genre. Sort the result based on the average publication year in ascending order.