1. **Associations**

Item 1: How to Effectively Shape the @OneToMany Association

* Mối quan hệ @OneToMany bidirectional là mối quan hệ phổ biến nhất trong mô hình miền.
* Luôn sử dụng mối quan hệ @OneToMany bidirectional thay vì unidirectional (một chiều).
* Luôn cascade từ phía cha đến phía con và không ngược lại. Điều này đảm bảo việc tạo ra con không tự động tạo ra lớp cha.
* Sử dụng "mappedBy" trên phía lớp cha để chỉ định mối quan hệ bidirectional và đồng bộ với phía con.
* Đặt orphanRemoval trên phía cha để đảm bảo việc xóa các con mà không có tham chiếu.
* Giữ cả hai phía của mối quan hệ đồng bộ nhau thông qua các phương thức trợ giúp trên phía cha.
* Ghi đè phương thức equals() và hashCode() đúng cách trên phía con.
* Sử dụng lazy fetching cho cả hai phía của mối quan hệ để tối ưu hóa hiệu suất.
* Sử dụng @JoinColumn để chỉ định tên cột tham gia trong mối quan hệ.

Item 2: Why You Should Avoid the Unidirectional @OneToMany

Association

* Trong liên kết @OneToMany hai chiều, việc chèn hoặc xóa một thực thể con (ví dụ: Sách) sẽ kích hoạt một câu lệnh SQL duy nhất đối với bảng sách. Tuy nhiên, trong liên kết @OneToMany một chiều, một bảng nối bổ sung được tạo để quản lý liên kết, dẫn đến tiêu thụ nhiều bộ nhớ hơn và có khả năng ảnh hưởng đến hoạt động truy vấn.
* Liên kết @OneToMany một chiều yêu cầu các câu lệnh SQL bổ sung để chèn và xóa dữ liệu. Khi duy trì một tác giả với các cuốn sách được liên kết, có các câu lệnh INSERT bổ sung trong bảng nối. Khi thêm một cuốn sách mới, những cuốn sách được liên kết hiện có sẽ bị xóa và sau đó được thêm lại, điều này không hiệu quả. Tương tự, khi xóa một cuốn sách, tất cả các cuốn sách được liên kết sẽ bị xóa khỏi bảng nối và sau đó được thêm lại, dẫn đến một hình phạt về hiệu suất.
* Bằng cách sử dụng chú thích @OrderColumn trong liên kết @OneToMany một chiều, bộ sưu tập sẽ được sắp xếp theo thứ tự và chỉ mục phần tử được lưu trữ trong một cột riêng của bảng liên kết. Tuy nhiên, điều này vẫn dẫn đến câu lệnh INSERT bổ sung cho sách mới và câu lệnh DELETE bổ sung khi xóa sách.
* Việc thêm chú thích @JoinColumn trong liên kết @OneToMany một chiều cho phép kiểm soát khóa ngoại của bảng con, loại bỏ sự cần thiết của bảng nối. Tuy nhiên, phương pháp này yêu cầu các câu lệnh UPDATE bổ sung để đặt các giá trị khóa ngoại, dẫn đến hình phạt về hiệu suất.
* Nhìn chung, việc sử dụng liên kết @OneToMany hai chiều hiệu quả hơn so với liên kết đơn hướng về đọc, ghi và xóa dữ liệu. Liên kết một chiều giới thiệu các câu lệnh SQL bổ sung và các hình phạt hiệu suất tiềm năng, đặc biệt là khi xử lý một số lượng lớn các thực thể được liên kết.

Item 3: How Efficient Is the Unidirectional @ManyToOne

* Mối quan hệ `@ManyToOne` không định hướng trong Hibernate có hiệu suất khá hiệu quả, và nó có thể được sử dụng khi không cần thiết mối quan hệ `@OneToMany` hai chiều. Trong ví dụ này, mối quan hệ giữa hai thực thể `Author` và `Book` được sử dụng, trong đó mỗi cuốn sách thuộc về một tác giả duy nhất.
* Khi thêm một cuốn sách mới cho một tác giả cụ thể, cách tiếp cận hiệu quả nhất là lấy tham chiếu đến thực thể `Author` bằng cách sử dụng `getOne()` (hoặc `findById()` nếu bạn cần tải thực thể cụ thể) và đặt nó làm tác giả của cuốn sách mới. Thao tác này sẽ kích hoạt một câu lệnh SQL `INSERT` duy nhất, điền giá trị của cột `author\_id` bằng mã định danh của thực thể `Author` tương ứng.
* Khi lấy tất cả các cuốn sách của một tác giả, bạn có thể thực hiện một truy vấn JPQL với điều kiện dựa trên ID của tác giả. Thao tác này sẽ kích hoạt một câu lệnh `SELECT` duy nhất, trả về các cuốn sách phù hợp.
* Khi sửa đổi thông tin một cuốn sách trong tập hợp đã lấy, Hibernate sử dụng cơ chế Dirty Checking để tự động kích hoạt câu lệnh `UPDATE`. Nếu bạn cập nhật các thuộc tính của một cuốn sách, Hibernate sẽ tự động gửi câu lệnh `UPDATE` tương ứng.
* Đối với việc phân trang, khi lấy một phần của tập hợp các cuốn sách, bạn có thể sử dụng phân trang với offset thông qua tham số `Pageable`. Phương pháp này thực hiện hai câu lệnh SELECT, một để lấy tập hợp giới hạn các cuốn sách và một để xác định tổng số cuốn sách.
* Khi lấy tất cả các cuốn sách của một tác giả và đồng thời thêm một cuốn sách mới, thao tác này vẫn được thực hiện hiệu quả. Thao tác lấy yêu cầu một câu lệnh `SELECT` duy nhất, và việc thêm một cuốn sách mới vào tập hợp đã lấy sẽ kích hoạt một câu lệnh `INSERT` duy nhất.
* Tương tự, khi lấy tất cả các cuốn sách của một tác giả và xóa một cuốn sách, quá trình vẫn hiệu quả. Thao tác lấy yêu cầu một câu lệnh `SELECT` duy nhất, và việc xóa một cuốn sách khỏi tập hợp đã lấy sẽ kích hoạt một câu lệnh `DELETE` duy nhất.
* Tổng quan, mối quan hệ `@ManyToOne` không định hướng trong Hibernate được chứng minh là hiệu quả và có thể được sử dụng khi không cần thiết mối quan hệ `@OneToMany` hai chiều. Tuy nhiên, để có hiệu suất tốt hơn, nên tránh sử dụng mối quan hệ `@OneToMany` không định hướng, như đã thảo luận trong Mục 2.

Item 4: How to Effectively Shape the @ManyToMany Association

* Sử dụng Set thay vì List: Để tối ưu việc xóa đối tượng, nên sử dụng Set thay vì List. Set có hiệu suất tốt hơn List.
* Tránh sử dụng CascadeType.ALL và CascadeType.REMOVE: Trong hầu hết các trường hợp, việc tự động xóa các đối tượng liên quan khi xóa một đối tượng cha có thể gây ra vấn đề. Trong trường hợp này, nên sử dụng CascadeType.PERSIST và CascadeType.MERGE để điều khiển cách cascade xảy ra.

Item 5: Why Set Is Better than List in @ManyToMany

* Sử dụng Set thay vì List trong quan hệ @ManyToMany: Khi xóa hoặc sắp xếp lại các đối tượng trong mối quan hệ, việc sử dụng Set sẽ giảm số lượng truy vấn SQL cần thực hiện. Set không bảo đảm thứ tự của các phần tử, nhưng trong trường hợp này, việc duy trì thứ tự không quan trọng.
* Tránh sử dụng CascadeType.ALL và CascadeType.REMOVE: Trong hầu hết các trường hợp, việc tự động xóa các đối tượng liên quan khi xóa một đối tượng cha có thể gây ra vấn đề. Trong trường hợp này, nên sử dụng CascadeType.PERSIST và CascadeType.MERGE để điều khiển cách cascade xảy ra.

Item 6: Why and When to Avoid Removing Child Entities with CascadeType.Remove and orphanRemoval=true

* CascadeType.REMOVE và orphanRemoval=true đều được sử dụng để tự động xóa các đối tượng con khi đối tượng cha bị xóa.
* orphanRemoval=true được sử dụng để xóa các tham chiếu lạc hậu (dangling references) tức là các đối tượng con mà không tồn tại mà không có tham chiếu từ đối tượng cha.
* CascadeType.REMOVE không cần thiết trong trường hợp ngắt kết nối một đối tượng con với đối tượng cha, vì nó không phải là một thao tác xóa.
* orphanRemoval=true có thể gây ra hiệu suất kém nếu áp dụng cho một số lượng lớn các đối tượng.
* Có các phương pháp tối ưu để xóa các đối tượng cha và con bằng cách sử dụng các thao tác tập trung (bulk operations) như deleteInBatch và deleteByIdentifier. Tuy nhiên, các phương pháp này có một số hạn chế như không hỗ trợ Optimistic Locking tự động và không tận dụng được tính năng cascading removals.
* Để đảm bảo Persistence Context được đồng bộ sau các thao tác xóa, cần sử dụng flushAutomatically = true và clearAutomatically = true hoặc gọi phương thức flush() và clear()

Item 7: How to Fetch Associations via JPA Entity Graphs

* Entity Graphs giúp định nghĩa các mối quan hệ liên quan đến thực thể và các trường cơ bản cần được tải trong một câu lệnh SELECT duy nhất. Đồng thời, chúng có thể được tái sử dụng và áp dụng cho các truy vấn khác nhau.
* Có hai cách để định nghĩa Entity Graphs:
  + Sử dụng chú thích @NamedEntityGraph trên thực thể và chỉ định các thuộc tính cần tải.
  + Sử dụng @EntityGraph annotation với attributePaths để tạo một Entity Graph tạm thời.

Item 8: How Efficient Is the Unidirectional @ManyToOne

Item 9: How Efficient Is the Unidirectional @ManyToOne