

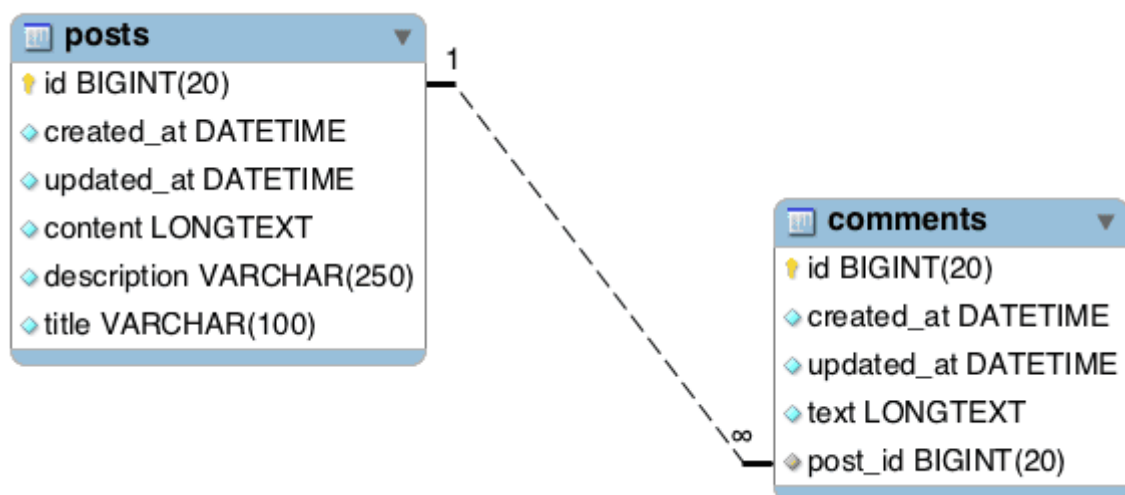
# JPA / Hibernate One to Many Mapping Example with Spring Boot

by RAJEEV SINGH · SPRING BOOT · NOVEMBER 24, 2017 · 5 MINS READ

## JPA / Hibernate One to Many Mapping

In this article, you'll learn how to map a one-to-many database relationship at the object level using JPA and Hibernate.

Consider the following two tables - `posts` and `comments` of a Blog database schema where the `posts` table has a one-to-many relationship with the `comments` table -



We'll create a project from scratch and learn how to go about implementing such one-to-many relationship at the object level using JPA and hibernate.

We'll also write REST APIs to perform CRUD operations on the entities so that you fully understand how to actually use these relationships in the real world.

## Creating the Project

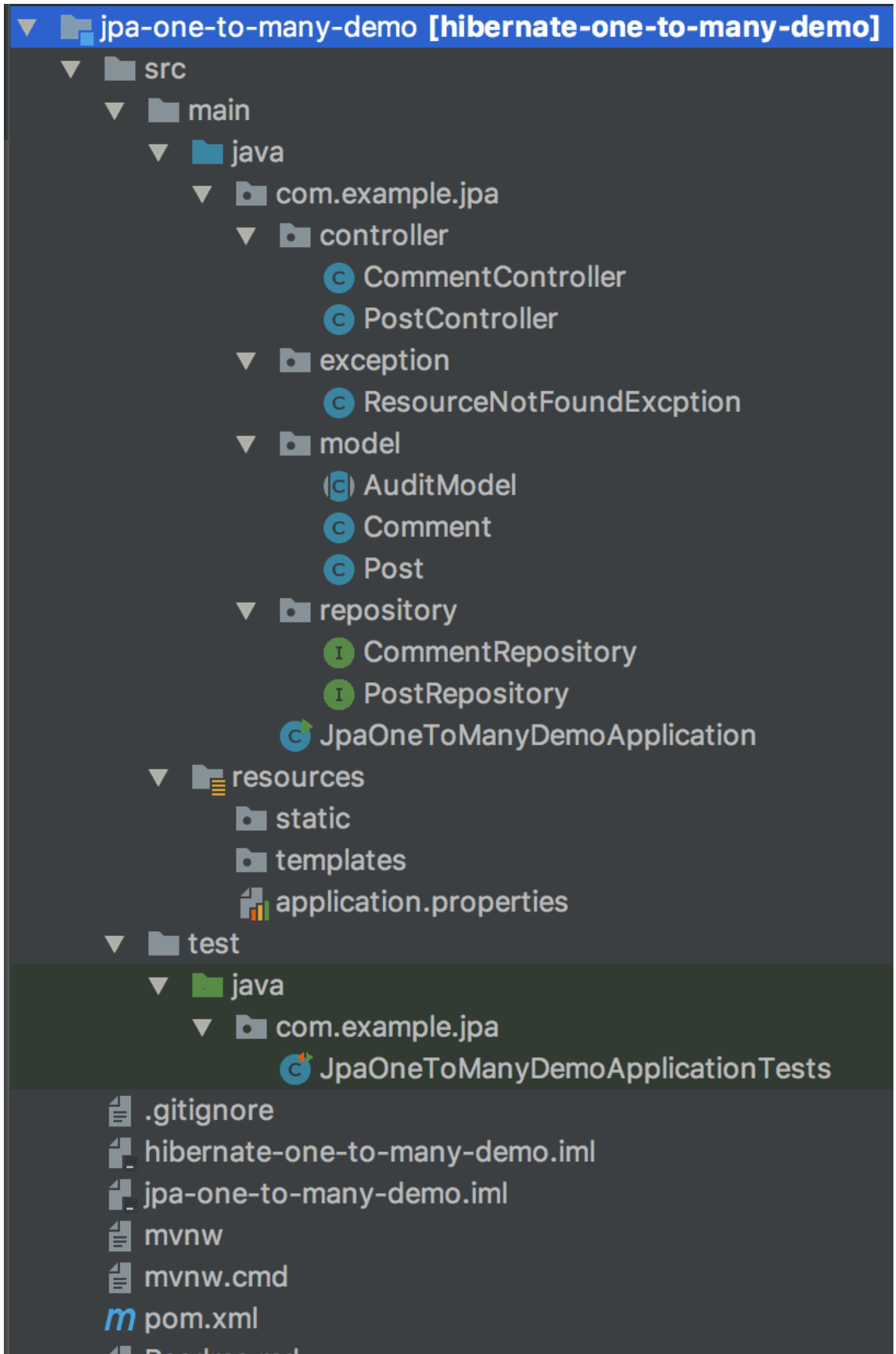
If you have [Spring Boot CLI](#) installed, then you can type the following command in your terminal to generate the project -

```
spring init -n=jpa-one-to-many-demo -d=web,jpa,mysql --package-name=com.e
```

Alternatively, You can generate the project from [Spring Initializr](#) web tool by following the instructions below -

1. Go to <http://start.spring.io>
2. Enter **Artifact** as "jpa-one-to-many-demo"
3. Click **Options** dropdown to see all the options related to project metadata.
4. Change **Package Name** to "com.example.jpa"
5. Select **Web**, **JPA** and **Mysql** dependencies.
6. Click **Generate** to download the project.

Following is the directory structure of the project for your reference -



packages, and all the classes inside these packages at this point. We'll create them shortly."

## Configuring the Database and Logging

Since we're using MySQL as our database, we need to configure the database URL, username, and password so that Spring can establish a connection with the database on startup. Open `src/main/resources/application.properties` file and add the following properties to it -

```
# DATASOURCE (DataSourceAutoConfiguration & DataSourceProperties)
spring.datasource.url=jdbc:mysql://localhost:3306/jpa_one_to_many_demo?use
spring.datasource.username=root
spring.datasource.password=root

# Hibernate

# The SQL dialect makes Hibernate generate better SQL for the chosen data
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Inno

# Hibernate ddl auto (create, create-drop, validate, update)
spring.jpa.hibernate.ddl-auto = update

logging.level.org.hibernate.SQL=DEBUG
logging.level.org.hibernate.type=TRACE
```

Don't forget to change the `spring.datasource.username` and `spring.datasource.password` as per your MySQL installation. Also, create a database named `jpa_one_to_many_demo` in MySQL before proceeding to the next section.

You don't need to create any tables. The tables will automatically be created by hibernate from the `Post` and `Comment` entities that we will define shortly. This is made possible by the property `spring.jpa.hibernate.ddl-auto = update`.

We have also specified the log levels for hibernate so that we can debug all the SQL statements and learn what hibernate does under the hood.

# The best way to model a one-to-many relationship in hibernate

I have been working with hibernate for quite some time and I've realized that **the best way to model a one-to-many relationship** is to use just `@ManyToOne` annotation on the child entity.

The second best way is to define a bidirectional association with a `@OneToMany` annotation on the parent side of the relationship and a `@ManyToOne` annotation on the child side of the relationship. The bidirectional mapping has its pros and cons. I'll demonstrate these pros and cons in the second section of this article. I'll also tell you when a bidirectional mapping is a good fit.

But let's first model our one-to-many relationship in the best way possible.

## Defining the Domain Models

In this section, we'll define the domain models of our application - `Post` and `Comment`.

Note that both `Post` and `Comment` entities contain some common auditing related fields like `created_at` and `updated_at`.

We'll abstract out these common fields in a separate class called `AuditModel` and extend this class in the `Post` and `Comment` entities.

We'll also use Spring Boot's [JPA Auditing](#) feature to automatically populate the `created_at` and `updated_at` fields while persisting the entities.

### 1. AuditModel

```
package com.example.jpa.model;

import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import org.springframework.data.annotation.CreatedDate;
import org.springframework.data.annotation.LastModifiedDate;
import org.springframework.data.jpa.domain.support.AuditingEntityListener;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import java.io.Serializable;
```

```
import java.util.Date;

@MappedSuperclass
@EntityListeners(AuditingEntityListener.class)
@JsonIgnoreProperties(
    value = {"createdAt", "updatedAt"},
    allowGetters = true
)
public abstract class AuditModel implements Serializable {
    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "created_at", nullable = false, updatable = false)
    @CreatedDate
    private Date createdAt;

    @Temporal(TemporalType.TIMESTAMP)
    @Column(name = "updated_at", nullable = false)
    @LastModifiedDate
    private Date updatedAt;

    public Date getCreatedAt() {
        return createdAt;
    }

    public void setCreatedAt(Date createdAt) {
        this.createdAt = createdAt;
    }

    public Date getUpdatedAt() {
        return updatedAt;
    }

    public void setUpdatedAt(Date updatedAt) {
        this.updatedAt = updatedAt;
    }
}
```

In the above class, we're using Spring Boot's `AuditingEntityListener` to automatically populate the `createdAt` and `updatedAt` fields.

## Enabling JPA Auditing

To enable JPA Auditing, you'll need to add `@EnableJpaAuditing` annotation to one of your configuration classes. Open the main class `JpaOneToManyDemoApplication.java` and add the `@EnableJpaAuditing` to the main class like so -

```
package com.example.jpa;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.data.jpa.repository.config.EnableJpaAuditing;

@SpringBootApplication
@EnableJpaAuditing
public class JpaOneToManyDemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(JpaOneToManyDemoApplication.class, args);
    }
}
```

## 2. Post model

```
package com.example.jpa.model;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

@Entity
@Table(name = "posts")
public class Post extends AuditModel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    @Size(max = 100)
```

```
@Column(unique = true)
private String title;

@NotNull
@Size(max = 250)
private String description;

@NotNull
@Lob
private String content;

// Getters and Setters (Omitted for brevity)
}
```

### 3. Comment model

```
package com.example.jpa.model;

import com.fasterxml.jackson.annotation.JsonIgnore;
import javax.persistence.*;
import javax.validation.constraints.NotNull;
import org.hibernate.annotations.OnDelete;
import org.hibernate.annotations.OnDeleteAction;

@Entity
@Table(name = "comments")
public class Comment extends AuditModel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    @Lob
    private String text;

    @ManyToOne(fetch = FetchType.LAZY, optional = false)
    @JoinColumn(name = "post_id", nullable = false)
```



```
@OnDelete(action = OnDeleteAction.CASCADE)
@JsonIgnore
private Post post;

// Getters and Setters (Omitted for brevity)
}
```

The `Comment` model contains the `@ManyToOne` annotation to declare that it has a many-to-one relationship with the `Post` entity. It also uses the `@JoinColumn` annotation to declare the foreign key column.

## Defining the Repositories

Next, We'll define the repositories for accessing the data from the database. Create a new package called `repository` inside `com.example.jpa` package and add the following interfaces inside the `repository` package -

### 1. PostRepository

```
package com.example.jpa.repository;

import com.example.jpa.model.Post;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface PostRepository extends JpaRepository<Post, Long> {
```

```
}
```

## 2. CommentRepository

```
package com.example.jpa.repository;

import com.example.jpa.model.Comment;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import java.util.Optional;

@Repository
public interface CommentRepository extends JpaRepository<Comment, Long> {
    Page<Comment> findByPostId(Long postId, Pageable pageable);
    Optional<Comment> findByIdAndPostId(Long id, Long postId);
}
```

## Writing the REST APIs to perform CRUD operations on the entities

Let's now write the REST APIs to perform CRUD operations on `Post` and `Comment` entities.

All the following controller classes are define inside `com.example.jpa.controller` package.

### 1. PostController (APIs to create, retrieve, update, and delete Posts)

```
package com.example.jpa.controller;

import com.example.jpa.exception.ResourceNotFoundException;
import com.example.jpa.model.Post;
```

```
import com.example.jpa.repository.PostRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

@RestController
public class PostController {

    @Autowired
    private PostRepository postRepository;

    @GetMapping("/posts")
    public Page<Post> getAllPosts(Pageable pageable) {
        return postRepository.findAll(pageable);
    }

    @PostMapping("/posts")
    public Post createPost(@Valid @RequestBody Post post) {
        return postRepository.save(post);
    }

    @PutMapping("/posts/{postId}")
    public Post updatePost(@PathVariable Long postId, @Valid @RequestBody
        return postRepository.findById(postId).map(post -> {
            post.setTitle(postRequest.getTitle());
            post.setDescription(postRequest.getDescription());
            post.setContent(postRequest.getContent());
            return postRepository.save(post);
        }).orElseThrow(() -> new ResourceNotFoundException("PostId " + postId))
    }

    @DeleteMapping("/posts/{postId}")
    public ResponseEntity<?> deletePost(@PathVariable Long postId) {
        return postRepository.findById(postId).map(post -> {
```

```

        postRepository.delete(post);
        return ResponseEntity.ok().build();
    }).orElseThrow(() -> new ResourceNotFoundException("PostId " + post.getId()));
}
}

```

## 2. CommentController (APIs to create, retrieve, update, and delete Comments)

```

package com.example.jpa.controller;

import com.example.jpa.exception.ResourceNotFoundException;
import com.example.jpa.model.Comment;
import com.example.jpa.repository.CommentRepository;
import com.example.jpa.repository.PostRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;

import javax.validation.Valid;

@RestController
public class CommentController {

    @Autowired
    private CommentRepository commentRepository;

    @Autowired
    private PostRepository postRepository;

    @GetMapping("/posts/{postId}/comments")
    public Page<Comment> getAllCommentsByPostId(@PathVariable (value = "postId") Long postId,
                                                Pageable pageable) {
        return commentRepository.findByPostId(postId, pageable);
    }
}

```

```
}
```

```
@PostMapping("/posts/{postId}/comments")
public Comment createComment(@PathVariable (value = "postId") Long postId,
                             @Valid @RequestBody Comment comment) {
    return postRepository.findById(postId).map(post -> {
        comment.setPost(post);
        return commentRepository.save(comment);
    }).orElseThrow(() -> new ResourceNotFoundException("PostId " + postId))
}
```

```
@PutMapping("/posts/{postId}/comments/{commentId}")
public Comment updateComment(@PathVariable (value = "postId") Long postId,
                             @PathVariable (value = "commentId") Long commentId,
                             @Valid @RequestBody Comment commentRequest) {
    if(!postRepository.existsById(postId)) {
        throw new ResourceNotFoundException("PostId " + postId + " not found")
    }

    return commentRepository.findById(commentId).map(comment -> {
        comment.setText(commentRequest.getText());
        return commentRepository.save(comment);
    }).orElseThrow(() -> new ResourceNotFoundException("CommentId " + commentId))
}
```

```
@DeleteMapping("/posts/{postId}/comments/{commentId}")
public ResponseEntity<?> deleteComment(@PathVariable (value = "postId") Long postId,
                                       @PathVariable (value = "commentId") Long commentId) {
    return commentRepository.findByIdAndPostId(commentId, postId).map(comment -> {
        commentRepository.delete(comment);
        return ResponseEntity.ok().build();
    }).orElseThrow(() -> new ResourceNotFoundException("Comment not found"))
}
```

## The ResourceNotFoundException Class

Both the Post and Comment Rest APIs throw `ResourceNotFoundException` when a post or comment could not be found. Following is the definition of the `ResourceNotFoundException`.

```
package com.example.jpa.exception;

import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ResponseStatus;

@ResponseStatus(HttpStatus.NOT_FOUND)
public class ResourceNotFoundException extends RuntimeException {
    public ResourceNotFoundException() {
        super();
    }

    public ResourceNotFoundException(String message) {
        super(message);
    }

    public ResourceNotFoundException(String message, Throwable cause) {
        super(message, cause);
    }
}
```

I have added the `@ResponseStatus(HttpStatus.NOT_FOUND)` annotation to the above exception class to tell Spring Boot to respond with a 404 status when this exception is thrown.

## Running the Application and Testing the APIs via a Postman

You can run the application by typing the following command in the terminal -

```
mvn spring-boot:run
```

Let's now test the APIs via Postman.

## Create Post POST /posts

The screenshot shows a REST client interface with the following details:

- URL:** http://localhost:8080/
- Method:** POST
- Path:** http://localhost:8080/posts/
- Environment:** No Environment
- Body Type:** raw (selected), JSON (application/json)
- Request Body:**

```
1 {  
2   "title": "Post 1",  
3   "description": "Post 1 description",  
4   "content": "Post 1 content"  
5 }
```
- Response Status:** 200 OK, Time: 105 ms
- Response Body (Pretty):**

```
1 {  
2   "createdAt": "2018-04-30T05:25:29.978+0000",  
3   "updatedAt": "2018-04-30T05:25:29.978+0000",  
4   "id": 8,  
5   "title": "Post 1",  
6   "description": "Post 1 description",  
7   "content": "Post 1 content"  
8 }
```

## Get paginated Posts GET /posts?

page=0&size=2&sort=createdAt,desc

http://localhost:8080/ No Environment

GET http://localhost:8080/posts?page=0&size=2&sort=createdAt,desc Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

Type No Auth

Body Cookies Headers (3) Test Results Status: 200 OK Time: 35 ms

Pretty Raw Preview JSON

```
1 {
2   "content": [
3     {
4       "createdAt": "2018-04-30T05:26:44.000+0000",
5       "updatedAt": "2018-04-30T05:26:44.000+0000",
6       "id": 10,
7       "title": "Post 3",
8       "description": "Post 3 description",
9       "content": "Post 3 content"
10    },
11    {
12      "createdAt": "2018-04-30T05:26:37.000+0000",
13      "updatedAt": "2018-04-30T05:26:37.000+0000",
14      "id": 9,
15      "title": "Post 2",
16      "description": "Post 2 description",
17      "content": "Post 2 content"
18    }
19  ],
20  "pageable": {
21    "sort": {
22      "sorted": true,
23      "unsorted": false
24    },
25    "offset": 0,
```

## Create Comment POST /posts/{postId}/comments

http://localhost:8080/ No Environment

POST http://localhost:8080/posts/8/comments Params Send Save

Authorization Headers (1) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary JSON (application/json)

```
1 {
2   "text": "Great Post"
3 }
```

Body Cookies Headers (3) Test Results Status: 200 OK Time: 147 ms

Pretty Raw Preview JSON

```
1 {
2   "createdAt": "2018-04-30T05:34:19.523+0000",
3   "updatedAt": "2018-04-30T05:34:19.523+0000",
4   "id": 1,
5   "text": "Great Post"
6 }
```

## Get paginated comments GET /posts/{postId}/comments? page=0&size=3&sort=createdAt,desc



http://localhost:8080/

GET http://localhost:8080/posts/8/comments?page=0&size=3&sort=createdAt,desc

Authorization Headers (1) Body Pre-request Script Tests

Type No Auth

Body Cookies Headers (3) Test Results Status: 200 OK Time: 909 ms

```

1 {
2   "content": [
3     {
4       "createdAt": "2018-04-30T05:35:03.000+0000",
5       "updatedAt": "2018-04-30T05:35:03.000+0000",
6       "id": 3,
7       "text": "It's not working for me"
8     },
9     {
10      "createdAt": "2018-04-30T05:34:53.000+0000",
11      "updatedAt": "2018-04-30T05:34:53.000+0000",
12      "id": 2,
13      "text": "Awesome Post. Thank you"
14    },
15    {
16      "createdAt": "2018-04-30T05:34:20.000+0000",
17      "updatedAt": "2018-04-30T05:34:20.000+0000",
18      "id": 1,
19      "text": "Great Post"
20    }
21  ],
22  "pageable": {
23    "sort": {
24      "sorted": true,
25      "unsorted": false
  
```

You can test other APIs in the same way.

## How to define a bidirectional one-to-many mapping and when should you use it

The Internet is flooded with examples of bidirectional one-to-many mapping. But it's not the best and the most efficient way to model a one-to-many relationship.

Here is the bidirectional version of the one-to-many relationship between the `Post` and `Comment` entities -

### Post Entity

```

package com.example.jpa.model;

import javax.persistence.*;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;
import java.util.*;
  
```

```
@Entity
@Table(name = "posts")
public class Post extends AuditModel {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @NotNull
    @Size(max = 100)
    @Column(unique = true)
    private String title;

    @NotNull
    @Size(max = 250)
    private String description;

    @NotNull
    @Lob
    private String content;

    @OneToMany(cascade = CascadeType.ALL,
              fetch = FetchType.LAZY,
              mappedBy = "post")
    private Set<Comment> comments = new HashSet<>();

    // Getters and Setters (Omitted for brevity)
}
```

## Comment Entity

```
package com.example.jpa.model;

import javax.persistence.*;
import javax.validation.constraints.NotNull;

@Entity
@Table(name = "comments")
```

```
public class Comment extends AuditModel {  
    @Id  
    @GeneratedValue(strategy = GenerationType.IDENTITY)  
    private Long id;  
  
    @NotNull  
    @Lob  
    private String text;  
  
    @ManyToOne(fetch = FetchType.LAZY)  
    @JoinColumn(name = "post_id", nullable = false)  
    private Post post;  
  
    // Getters and Setters (Omitted for brevity)  
}
```

The idea with bidirectional one-to-many association is to allow you to keep a collection of child entities in the parent, and enable you to persist and retrieve the child entities via the parent entity. This is made possible via Hibernate's [entity state transitions](#) and [dirty checking mechanism](#).

For example, here is how you could persist comments via `post` entity in the bidirectional mapping -

```
// Create a Post  
Post post = new Post("post title", "post description", "post content");  
  
// Create Comments  
Comment comment1 = new Comment("Great Post!");  
comment1.setPost(post);  
Comment comment2 = new Comment("Really helpful Post. Thanks a lot!");  
comment2.setPost(post);  
  
// Add comments in the Post  
post.getComments().add(comment1);  
post.getComments().add(comment2);
```

```
// Save Post and Comments via the Post entity
postRepository.save(post);
```

Hibernate automatically issues insert statements and saves the comments added to the post.

Similarly, you could fetch comments via the `post` entity like so -

```
// Retrieve Post
Post post = postRepository.findById(postId)

// Get all the comments
Set<Comment> comments = post.getComments()
```

When you write `post.getComments()`, hibernate loads all the comments from the database if they are not already loaded.

## Problems with bidirectional one-to-many mapping

- A bidirectional mapping tightly couples the many-side of the relationship to the one-side.
- In our example, If you load comments via the `post` entity, you won't be able to limit the number of comments loaded. That essentially means that you won't be able to paginate.
- If you load comments via the `post` entity, you won't be able to sort them based on different properties. You can define a default sorting order using `@OrderColumn` annotation but that will have performance implications.
- You'll find yourself banging your head around something called a `LazyInitializationException`.

## When can I use a bidirectional one-to-many mapping

A bidirectional one-to-many mapping might be a good idea if the number of child entities is limited.

Moreover, A bidirectional mapping tightly couples the many-side of the relationship to the one-side. *Many times, this tight coupling is desired.*

For example, Consider a Survey application with a `Question` and an `Option` entity exhibiting a one-to-many relationship between each other.

In the survey app, A `Question` can have a set of `Options`. Also, every `Question` is tightly coupled with its `Options`. When you create a `Question`, you'll also provide a set of `Options`. And, when you retrieve a `Question`, you will also need to fetch the `Options`.

Moreover, A `Question` can have at max 4 or 5 `Options`. These kind of cases are perfect for bi-directional mappings.

So, To decide between bidirectional and unidirectional mappings, you should think whether the entities have a tight coupling or not.

## Conclusion

That's all folks! In this article, You learned how to map a one-to-many database relationship using JPA and Hibernate.

You can find the code for the sample project that we built in this article in my [jpa-hibernate-tutorials repository](#) on github.

You might also be interested in checking out the following articles on JPA/Hibernate Mapping -

[JPA / Hibernate One to One mapping Example with Spring Boot](#)

[JPA / Hibernate Many to Many mapping Example with Spring Boot](#)

## More Resources

You may wanna check out the following articles by [Vlad Mihalcea](#) to learn more about Hibernate and it's performance -

- [The best way to map a one-to-many association with JPA and Hibernate](#)
- [14 high performance Java persistence tips](#)

Thanks for reading. I hope you found the content useful. Consider subscribing to my newsletter if you don't wanna miss future articles from The CalliCoder Blog.

---

**Share on social media**

---

Sponsored

**Die Schweiz staunt: Hörsystem, das sich jeder leisten kann!**

Hören heute

**Wenn du eine Maus hast, spiel es für 1 Minute und sieh warum jeder verrückt danach ist.**

CombatSiege

**Blaue Pille mischt Liebesleben auf: so einfach kaufst du die blaue Pille heute. (Gratis Lieferung)**

Mensmagazine

**Darmexperte bittet: "Lassen Sie Ihr Müsli weg, nehmen Sie stattdessen jeden Morgen das"**

Nutravia

**Mit diesem Trick können Sie Hörgeräte zum Nulltarif bekommen**

Hörgeräte Vergleich

**Wenn du eine Maus besitzt musst du unbedingt dieses Spiel ausprobieren.**

PlanetCapture

111 Comments

 Login ▼

G

Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS  15

Share

Best Newest Oldest

M

**Mondi Koci**

a year ago

Finally, it took me 4 days to land to this. Now I can move on. Thanks dude, just an awesome tutorial.

0 0 Reply • Share ›

**KRISTI JAN**

2 years ago

You'll find yourself banging your head around something called a LazyInitializationException.

Do you care to explain how to avoid it?

0 0 Reply • Share ›

N

**nihat**

2 years ago

Great tutorial but why you pass the entity direct to controller. Why you don't used for this purpose DTO pattern. By the way how can i learn deep dive this topic for example DTO? and how to pass dto to controller??

0 0 Reply • Share ›

N

**Noel Ajay**

2 years ago edited

How can I do the same in MVC and check results in JSP instead of postman. thanks.

0 0 Reply • Share ›

L

**LongMoon**

3 years ago

Hi, thank you for the wonderful article! This was very helpful to me!  
But I'm a beginner of Java & Spring Boot programming so I don't know how to connect these data to html(View) files.



Sponsored I searched a lot but I couldn't find any good examples....

Wenn du eine Maus hast, spiele es für 1 Minute und sieh warum jeder verrückt danach ist.

CombatSiege

0 0 Reply • Share ›

**K** **Kaleemullah Nizamani**

**Blau Pille mischt Liebesleben auf: so einfach kaufst du die blaue Pille heute. (Gratis Lieferung)**

Mensmagazine

Wow brother you cleared some of my concepts regarding uni and bi-directional relationship and their use cases. your code is always magic always works. :)

**Die Schweiz staunt: Hörsystem, das sich jeder leisten kann!**

Hören heute

One additional thing i would request you is explain the core concept in little more details for example

```
@ManyToOne(fetch = FetchType.LAZY, optional = false)
```

```
@JoinColumn(name = "post_id", nullable = false)
```

```
@OnDelete(action = OnDeleteAction.CASCADE)
```

**Darmexperte bittet: "Lassen Sie Ihr Müsli weg, nehmen Sie stattdessen jeden Morgen das"**

@Sonnig

each of above annotation and property requires some elaboration the reason i am focusing on

Nutravia

this code is it is core concept you are explaining mainly @ManyToOne uni-directional relationship.

**Ärzte verbieten: Ein einfacher Tipp gegen Nagelpilz (Heute Abend testen)**

PilzPlus+

0 0 Reply • Share ›

**W** **Winner** **Mit diesem Trick können Sie Hörgeräte zum Nulltarif bekommen**

Hörgeräte Vergleich

Hi... I would like to add many comment with List(List<comment> ... I am doing this but there is an error ... help me please... thank's

```
@PostMapping("/posts/{postId}/comments")
public List<comment> createComment(@PathVariable (value = "postId") Long
postId,
@Valid @RequestBody List<comment> comment) {
return postRepository.findById(postId).map(post -> {
comment.setPost(post); //error here
List<comment> response = (List<comment>)commentRepository.saveAll(comment);
return response;
}).orElseThrow(() -> new ResourceNotFoundException("PostId " + postId + " not
found"));
}
```

0 0 Reply • Share ›

**W** **Winner**

3 years ago edited

Hi; great tuto!! For @PostMapping; if we want to post more How at the same time how a List ... how we do!

how do i do this but there is an error !!

please help me

```
@PostMapping("/posts/{postId}/comments")
```

```

public List<comment> createComment(@PathVariable (value = "postId") Long postId,
@Valid @RequestBody List<comment> comment) {
return postRepository.findById(postId).map(post -> {
comment.setPost(post); //error here
List<comment> response = (List<comment>)commentRepository.saveAll(comment);
return response;
}).orElseThrow(() -> new ResourceNotFoundException("PostId " + postId + " not
found"));
}

```

0 0 Reply • Share ›

D

Diana Devasia

3 years ago

Hi @Rajeev Singh

I have followed your post for unidirectional manyToOne mapping. So when i try to get all the comments for a given post id , i still get LazyInitializationException.

Can you please help me.

@Entity

@Table(name = "post")

public class Post {

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE)

public Long id;

public String content

}

@Entity

@Table(name = "comment")

public class Comment {

@Id

@GeneratedValue(strategy = GenerationType.SEQUENCE)

see more

0 0 Reply • Share ›

K

Kaleemullah Nizamani

➔ Diana Devasia

3 years ago

use @JsonIgnore annotation on many to one side

0 0 Reply • Share ›

V

vinod patil

3 years ago

Hi Bro..Thank you so much for great help but I am getting below error while running the application

V

**www.pati**

3 years ago

Issue is solved now...Thank you bro for such great tutorial

0 0 Reply • Share ›



## Antoon Vereecken

3 years ago

Thank u sir <3

0 0 Reply • Share ›

S

**Siva**

3 years ago

i have four pojo classes with different fields and four repositories for that pojoes and i have only one controller class with only one @PostMapping. can i get all four classes of fields by using one PostMapping. Is it possible? explain me....

0 0 Reply • Share ›



## Sathish Kannan

3 years ago    edited

Very good article and crystal clear. Do you have any article that i can use JpaRepository and Native Query using @Query instead of Hibernate ?

0 0 Reply • Share ›

**X**

**xyzname myname**

3 years ago

Hi, Thank you so much for the article.

Is it possible to save data in both Post and Comment with one payload.

I tried with this, but it throws me error.

I modified the Post model class to this as mentioned in the article.

```
@OneToMany(fetch=FetchType.LAZY, mappedBy="post", cascade = CascadeType.ALL)
private List<comment> comment;
```

http://localhost:8080/posts

```
{
  "title": "Title4",
  "description": "Description of Title4",
  "content": "contentOfTitle4",
  "timestamp": 1
}
```

```
comment: [
  { "text": "This is Comment1 of 4"},
  { "text": "This is Comment2 of 4"}
]
```

"could not execute statement; SQL [n/a]; constraint [null]; nested exception is org.hibernate.exception.ConstraintViolationException: could not execute statement",

0 0 Reply • Share ›



**Shridhar Rao**

3 years ago

Thanks for the article.

0 0 Reply • Share ›



**Chris Allen Barroso**

3 years ago

Thank u for your fantastic job here. I love all your content.

I have try to implment OneToMany but am getting **Could not write JSON document: Infinite recursion**

Can u help me?

2019-10-28 18:17:48.755 WARN 18428 --- [nio-8080-exec-1]

.m.m.a.ExceptionHandlerExceptionResolver : Resolved exception caused by Handler execution: org.springframework.http.converter.HttpMessageNotWritableException: Could not write JSON document: Infinite recursion (StackOverflowError) (through reference chain: st.gov.financas.impostosApi.model.TbPedidoEmissaoDocumento["tbPedidoEmissaoFatura"]->st.gov.financas.impostosApi.model.TbPedidoEmissaoFatura["pedidoEmissaoDocumentos"]->org.hibernate.collection.internal.PersistentSet[0]-

1 0 Reply • Share ›



**Dr. Buffalo**

➔ Chris Allen Barroso

3 years ago edited

The "@JsonIgnoreProperties" annotation might help. For example in this tutorial, the author used

```
@JsonIgnoreProperties(value = {"createdAt", "updatedAt"}, allowGetters = true)
```

This prevents variables "createdAt" and "updatedAt" from being recursively printed out in the JSON output.

1 0 Reply • Share ›



**B. M. Shams Nahid**

4 years ago

Good one, thank you for you time.

2 0 Reply • Share ›

S

**Suhas Ramesh**

4 years ago edited

What is the purpose of using **findByldAndPostId** in the deleteComment function ? you could have found the comment by id itself and have deleted it, what is the use of findByldAndPostId ?

0 0 Reply • Share ›

V

**Vid tha**

4 years ago

is it possible to enter same data(all fields) into another table? i mean 2 tables have exact same data.

0 0 Reply • Share ›

C

**Cosmin**

4 years ago

Jesus Christ you freaking saved me, wasted like 16 hours to create this stupid relation and it seems I was too stupid to be able to create this stupid controller. I would make wild love to you if we were friends.

0 0 Reply • Share ›

R

**Ruchira Peiris**

4 years ago

Thank you very much for sharing Rajeev. When I'm trying to run the application it says, `findByPostId(java.lang.Integer,org.springframework.data.domain.Pageable)!` No property `postId` found for type `Comment!`. Could you suggest a solution for this?

1 1 Reply • Share ›

S

**Sree**

4 years ago edited

Thank you for Sharing

0 0 Reply • Share ›

**Awwal Fahru**

4 years ago

Thank you, Rajeev for sharing.. But I have a problem.. You said on [Comment.java](#)

private Post post;

But why on Comment Repository you write,

Page<comment> findByPostId (postId, pageable)

??

0 0 Reply • Share ›

**Amjith Titus**

➔ Awwal Fahru



4 years ago

What is your doubt exactly?

Pagination is used since there can be many comments.

Each comment is linked to a Post.

findByPostId is to find the comments for a post, hence it uses postId(Primary key of Post).

0 0 Reply • Share ›

**Gert Myburgh**

4 years ago

Once again, thank you very much for another great tutorial.

0 0 Reply • Share ›

**D****Devendra Choudhary**

4 years ago

Hi Rajeev,

Thanks for the great article. I am quite following your article and they are so nicely written.

In this article, we have used AuditModel Entity to keep track of createdAt and updatedAt dates. I wanted to go one step further and add createdBy and lastModifiedBy into AuditModel. I followed some other article for this also but I was facing problem with configuring properties.

I would be great if you can write a next part of this article where we have those two fields also in our AuditModel. Thanks in Advance and keep writing great article like this.

0 0 Reply • Share ›

**R****ranjesh**

4 years ago

I see a problem with unidirectional @ManyToOne, when parent is deleted , child is not deleted. I know you have used @OnDelete(action = OnDeleteAction.CASCADE) on child, But it does not work, ie when Post is deleted, comment is not deleted

0 0 Reply • Share ›

**S****SIIBERAD**

4 years ago

Thanks dude.. nice post :)

0 0 Reply • Share ›

**E****ele243**

4 years ago

In PostController, why is PostRepository not private? But in CommentController, CommentRepository is private?

Any specific reason for this difference?

0 0 Reply • Share ›

**Rajeev Singh** Mod

→ ele243



4 years ago

Hi,

Both should be private. I've missed adding `private` modifier to `PostRepository`. I'll fix that.

Thanks

1 0 Reply • Share ›

**Jack Rosios**

4 years ago

Hi there. Regarding to the `deleteComment()` method, the comment that exist will be deleted even though it is not relating to the parent post? Is it one of the disadvantage of non bidirectional?

0 0 Reply • Share ›

**E****ele243** → Jack Rosios

4 years ago

I am not an expert, so please ignore me if this wouldn't work.

Instead of using `comment.findById` and deleting it, what if we use `comment.findByPostIdAndId` or something like that so that we will be sure that the comment id {id} that we will be deleting belongs to post id {postId}.

0 0 Reply • Share ›

**Rajeev Singh** Mod

→ ele243



4 years ago

Hi,

That's an excellent point. It would be better to find the comment based on both `postId` and `Id`, and then delete it.

I'm gonna fix this.

Thanks

1 0 Reply • Share ›

**G****Guoliang Wang**

4 years ago

Why set the "unique = true" for the "title" property of `Post`. Title of post can be the same.  
Thank you

0 0 Reply • Share ›

**E****ele243** → Guoliang Wang

4 years ago

True, I think it was just a design decision that callicoder took. In real application, I too would want title to not be unique.

0 0 Reply • Share ›

S

**Seenivasan Sankaran**

4 years ago edited

Hi Rajeev,


I'm using Bidirectional Mapping, because my child entities are limited only.

My scenario


In Bidirectional mapping, I wrote one rest call to get all the post with appropriate comments for the post Using default findall method in JPA Repository . But the following exception occurred:

"message": "Could not write JSON: failed to lazily initialize a collection of role: com.insights.apartmento.modal.Post.comments, could not initialize proxy - no Session; nested exception is com.fasterxml.jackson.databind.JsonMappingException: failed to lazily initialize a collection of role: com.insights.apartmento.modal.Post.comments, could not initialize proxy - no Session (through reference chain: java.util.ArrayList[0]->com.insights.apartmento.modal.Post[\"comments\"])",

Post Pojo clas:

 [View](#) – uploads.disquscdn.com

Comment Pojo class:

 [View](#) – uploads.disquscdn.com

I want the output as follows:  [View](#) – uploads.disquscdn.com

Please help me...

Thanks in advance.

0 0 Reply • Share ›

C

**Chris**

4 years ago

Hi Rajeev,

instead making the ids like 1,2,3,4, is there a way to make them long uniquely generated Ids?

1 0 Reply • Share ›

J

**JollyRogerQZR**

5 years ago edited

Hey Rajeev,

I have a question pertaining this service. Suppose I need to get Post Id with every comment when I get Json response, so I don't get lost as to what comment belongs to what article when I get all. So @JsonIgnore is not an option, but I also don't want to get the content of the post - just id and maaaaybe name. What's the best way to do it?

I've tried Projections of Spring, didn't work.



I've tried Projections of Spring - didn't work;

potentially I can write another class "CommentWithPostId" and @Query there what I need but that's incredibly awkward.

Also I can transform Comment to CommentWithPostId, but to do that I'll need to fetch Post to every Comment, and I don't want to do that. In fact, I want Spring Data magic to only fetch the foreign key(that is a post Id) without going to Post table at all ideally.

@JsonView doesn't help either.

Thanks!

0 0 Reply • Share ›



**Rajeev Singh** Mod

→ JollyRogerQZR



5 years ago edited

Hi,

I've already answered a similar question in [this comment](#). Please check that out.

Cheers,  
Rajeev

0 0 Reply • Share ›

V

**Veda Gangatkar**



5 years ago

Hi Rajeev,

Have you written any JUnit or Mockito test cases for testing the end points?

0 0 Reply • Share ›



**vandu**



5 years ago

Great Tutorial!

I tried the 1st approach

Did i miss anything? I got the below error:

org.hibernate.MappingException: Could not determine type for: java.util.Set, at table: faqs, for columns: [org.hibernate.mapping.Column(comments)]

Please suggest. Thank you in advance.

Regards,  
Marimuthu

0 0 Reply • Share ›



**Rajeev Singh** Mod

→ vandu



5 years ago

Hi,

Looks like you've created some new classes for FAQs. Can you share your Entity class

definitions?

It's difficult to say anything without looking at the code.

0 1 Reply • Share ›



**Raushan Kumar Singh**

5 years ago

Awesome :)

0 0 Reply • Share ›



**Vương Khánh**

5 years ago edited

great tutorial <3

Thank you so much, this helps me a lot.

0 0 Reply • Share ›



**Tatiana**

5 years ago edited

Posts and comments do not have "a one-to-many relationship between each other." The relationship is one-to-many on the posts side and many-to-one on the comments side. You should probably be more specific about how you choose which side of the relationship you want to use as your label (because you could model either or both).

When you say it is one-to-many, you are only talking about the target side (the side that does not own the foreign key). If you don't say why you have chosen to talk about that particular side only, then it seems a bit arbitrary to call this simply a one-to-many relationship.

0 0 Reply • Share ›



**Rajeet Singh** Mod

➔ Tatiana

5 years ago

Hi,

You're right. I should re-phrase the first paragraph to better communicate the relationship between the two entities. I'll do that.

Thanks,  
Rajeet