Programming 5

JPA best practices

Karel de Grote Hogeschool

- JPA fetch type
- Query logging
- Optimizing query count
- Open session in view
- Summary

JPA fetch type

- The fetch type in JPA determines **when** a relationship is fetched.
- Lazy loading will gather the parent's child entities whenever it's needed. And when is that?
- Eager loading will always gather the parent's child entities as soon as the parent entity is loaded.

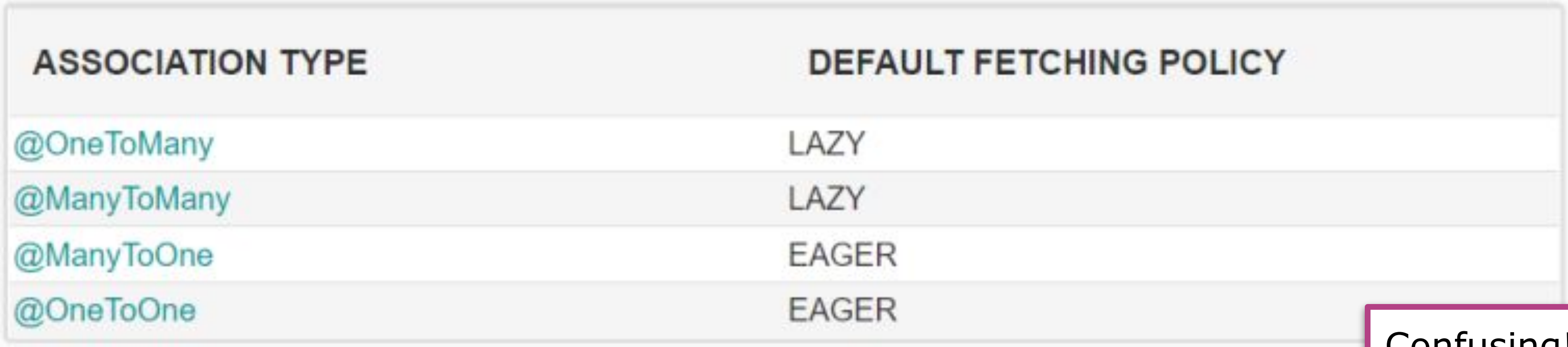
JPA fetch type

✓ **Guideline**: always use **the lazy fetch type**

- Once a relationship is set to be eagerly fetched, it cannot be changed to being fetched lazily on a per-query basis.
- Each business use case has different entity load requirements and therefore the fetching strategy should be delegated to each individual query.

The fetch type in JPA does **not** determine whether a SQL JOIN operation is used or whether an additional query is triggered.

Default fetching policies



ASSOCIATION TYPE	DEFAULT FETCHING POLICY
@OneToMany	LAZY
@ManyToMany	LAZY
@ManyToOne	EAGER
@OneToOne	EAGER

Confusing!

✓ **Guideline**: always set the fetch type explicitly

```
@OneToMany(mappedBy = "post", fetch = FetchType.LAZY)
private List<Comment> comments = new ArrayList<>();
```

- JPA fetch type
- Query logging
- Optimizing query count
- Open session in view
- Summary

Query logging

If you’re not careful, lazy loaded relationships might trigger lots of queries.

✓ **Guideline**: keep track of the amount of queries by turning on query logging

```
logging:
 level:
   "org.hibernate.SQL": DEBUG
   "org.hibernate.type.descriptor.sql.BasicBinder": TRACE
```

- JPA fetch type
- Query logging
- Optimizing query count
- Open session in view
- Summary

Optimizing query count

If you’re not careful, lazy loaded relationships might trigger lots of queries.

✓ **Guideline**: if you need extra data, write a custom query with `JOIN FETCH` instead of relying on lazy loading

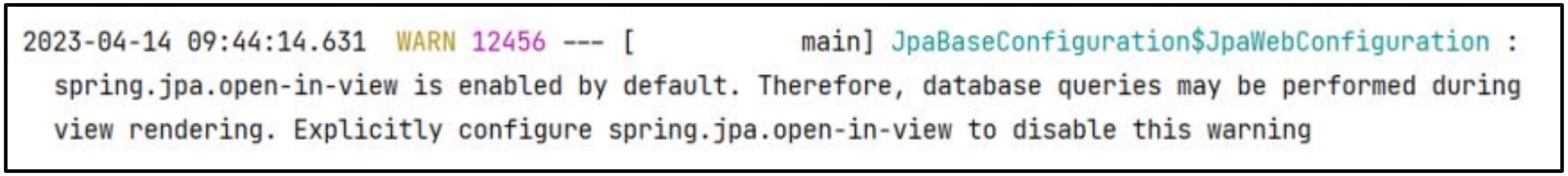
```
public interface QuestionRepository
        extends JpaRepository<Question, Long> {
    @Query("select question from Question question "
        + "left join fetch question.answers answers "
        + "where question.id = :questionId")
    Optional<Question> findWithAnswers(long questionId);
```

✓ **Guideline**: avoid lazy loading collections unless absolutely necessary

- JPA fetch type
- Query logging
- Optimizing query count
- Open session in view
- Summary

Open session in view

By default, Spring keeps a database session open during the entirety of a HTTP request.



```
2023-04-14 09:44:14.631  WARN 12456 --- [           main] JpaBaseConfiguration$JpaWebConfiguration :
  spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during
  view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
```

Calling a lazy loaded collection does not open a database session automatically. So, by enabling Open Session in View Spring ensures that you can fetch lazy loaded relationships from **any** point in the lifecycle of a HTTP request.

Open session in view on (default)

```
spring.jpa.open-in-view=true
```

```
@RestController
@RequestMapping("/issues")
public class IssueController {
    @GetMapping
    public String listIssues() {
        final List<Issue> issues = issueRepository.findAll();
        final StringBuilder sb = new StringBuilder();
        for (final Issue issue : issues) {
            sb.append(issue.getTitle());
            sb.append(issue.getAssignments().size() +
                " developers assigned");
        }
        return sb.toString();
    }
}
```

1: A database session is opened automatically when the request starts.

2: Calling `findAll()` on a repository method (or any method annotated with `@Transactional`) will open a database session if one is not active yet. In this case one is active already.

3: Loading a lazy collection will trigger an additional query to the database, and needs a database session to be active. In this case one is active already.

4: The database session is closed at the end of the request.

Open session in view off

```
spring.jpa.open-in-view=false
```

```
@RestController
@RequestMapping("/issues")
public class IssueController {
    @GetMapping
    public String listIssues() {
        final List<Issue> issues = issueRepository.findAll();
        final StringBuilder sb = new StringBuilder();
        for (final Issue issue : issues) {
            sb.append(issue.getTitle());
            sb.append(issue.getAssignments().size() +
                " developers assigned");
        }
        return sb.toString();
    }
}
```

1: A database session is **no longer** opened when the request starts.

2: Calling `findAll()` on a repository method (or any method annotated with `@Transactional`) will open a database session if one is not active yet. In this case it **will** open a database session.

3: Loading a lazy collection will trigger an additional query to the database, and needs a database session to be active. In this case, there is no session active and will throw a **`LazyInitializationException`**!

Solution to `LazyInitializationException`

To avoid `LazyInitializationException` when open session in view is **off**, you can do 2 things:

1. Avoid the additional query in the first place by using `JOIN FETCH`.

2. If lazy loading is really necessary, then ensure that a database session is active when you load the lazy relationship.

   Use `@Transactional`

It is always a good idea to be explicit about your transactional boundaries, so **always use this annotation <u>on the Service Layer</u>**, even when not strictly necessary.

Open session in view

- We do **not** want to be able to trigger database queries from any point in our application, but only from our service layer.
- We do **not** want to keep a database session open for longer than necessary.

✓ **Guideline**: turn open session in view **off** and add `@Transactional` to your service layer to ensure that a database session is only active for your business logic

```
spring.jpa.open-in-view=false
```

Baeldung

- JPA fetch type
- Query logging
- Optimizing query count
- Open session in view
- **Summary**

Summary

- ✓ **Guideline**: always use **the lazy fetch type**
- ✓ **Guideline**: always set the fetch type **explicitly**
- ✓ **Guideline**: if you need extra data, write a custom query with `JOIN FETCH` instead of relying on lazy loading
- ✓ **Guideline**: **avoid lazy loading collections** unless absolutely necessary
- ✓ **Guideline**: keep track of the amount of queries by turning on **query logging**
- ✓ **Guideline**: turn open session in view **off** and add `@Transactional` to your service layer to ensure that a database session is only active for your business logic

For the project, make sure you **understand** and **apply** these guidelines!