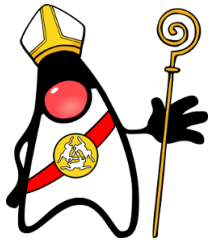
Building Fast and Scalable Persistence Layers with Spring Data JPA

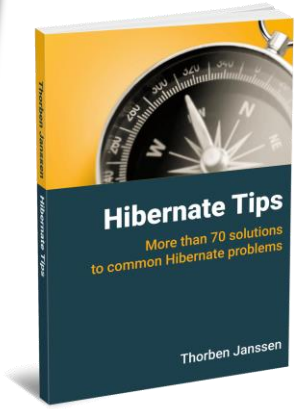
Thorben Janssen

Independent consultant, trainer and author



Co-Organizer
JUG Paderborn

thorben-janssen.com



Hibernate Tips
More than 70 solutions
to common Hibernate problems

www.hibernate-tips.com

@thjanssen123

/c/ThoughtsOnJava

thorbenjanssenofficial

Fast & Scalable

Performance

- Find problems as early as possible
- Avoid / Fix performance problems

Performance

- Persistence provider: Hibernate
  - Follow best practices
  - Avoid pitfalls

Identify Issues

Hibernate Statistics

- Activate via system property
  - hibernate.generate_statistics = true
- Configure logging
  - org.hibernate.stat = DEBUG

Slow Query Log

- Configure threshold
  - hibernate.session.events.log.LOG_QUERIES_SLOWER_THAN_MS

Code Samples

Association Fetching

FetchType

- Defines when the relationship will be fetched
- Static definition in entity mapping

```
@ManyToOne(fetch = FetchType.LAZY)
private Publisher publisher;
```

FetchType

- Lazy
  - Relationship gets loaded at first access
  - Default for to-many relationships
- Eager
  - Loads relationships immediately
  - Default for to-one relationships

Recommendation

- To-many relationships
  - Stick to the default mapping (FetchType.LAZY)
  - Use eager fetching for specific queries, if required
- To-one relationships
  - Check existing mappings individually
  - Use FetchType.LAZY for new ones

Query-Specific Fetching

N+1 Select?

- Most common problems
  - Lazy fetching of related entities

```
List<Author> authors = this.em.createQuery("SELECT a FROM Author a",
                                                    Author.class).getResultList();

for (Author a : authors) {
        System.out.println("Author " + a.getFirstName() + " " + a.getLastName()
                                + " wrote " + a.getBooks().size() + " Books."));
}
```

Query-Specific fetching

- Fetch all required entities with one query
  - Fetch Joins
  - EntityGraph

Code Samples

Many-to-Many Association

Many-to-Many

- Inefficient handling of List
  - Remove all associations
  - Add remaining ones
- Use Set instead

Code Samples

Projections

Projections

- Entities
- Scalar values
- DTO

DTO Projections

- DTO classes
- DTO interfaces
  - Spring Data JPA generates class

Code Samples

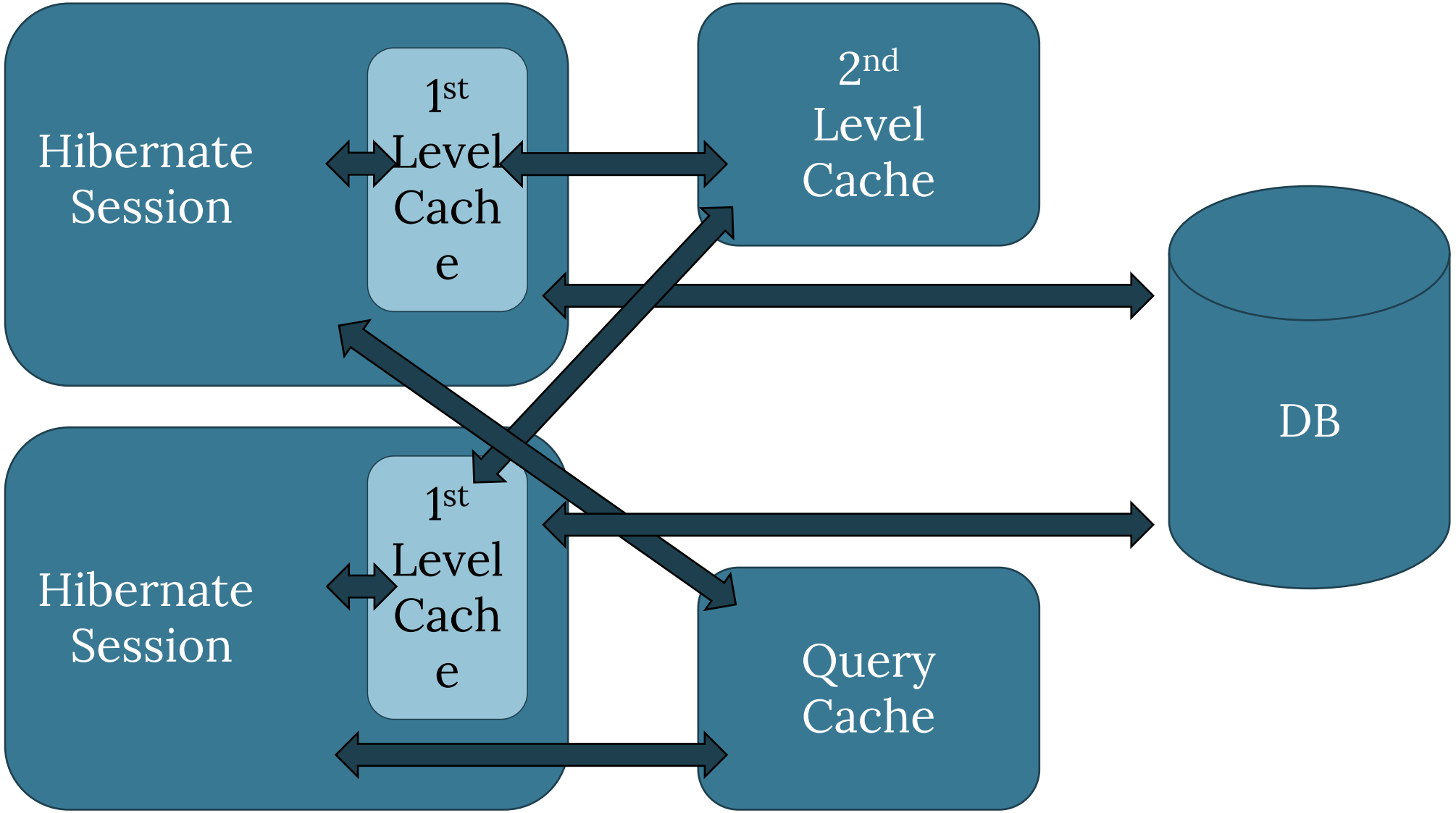
Advanced DTO Projections

- Nested associations
- Spring Expression Language

Code Samples

Caching

Caching


Hibernate Session
1st Level Cache
Hibernate Session
1st Level Cache
2nd Level Cache
Query Cache
DB

2nd Level Cache

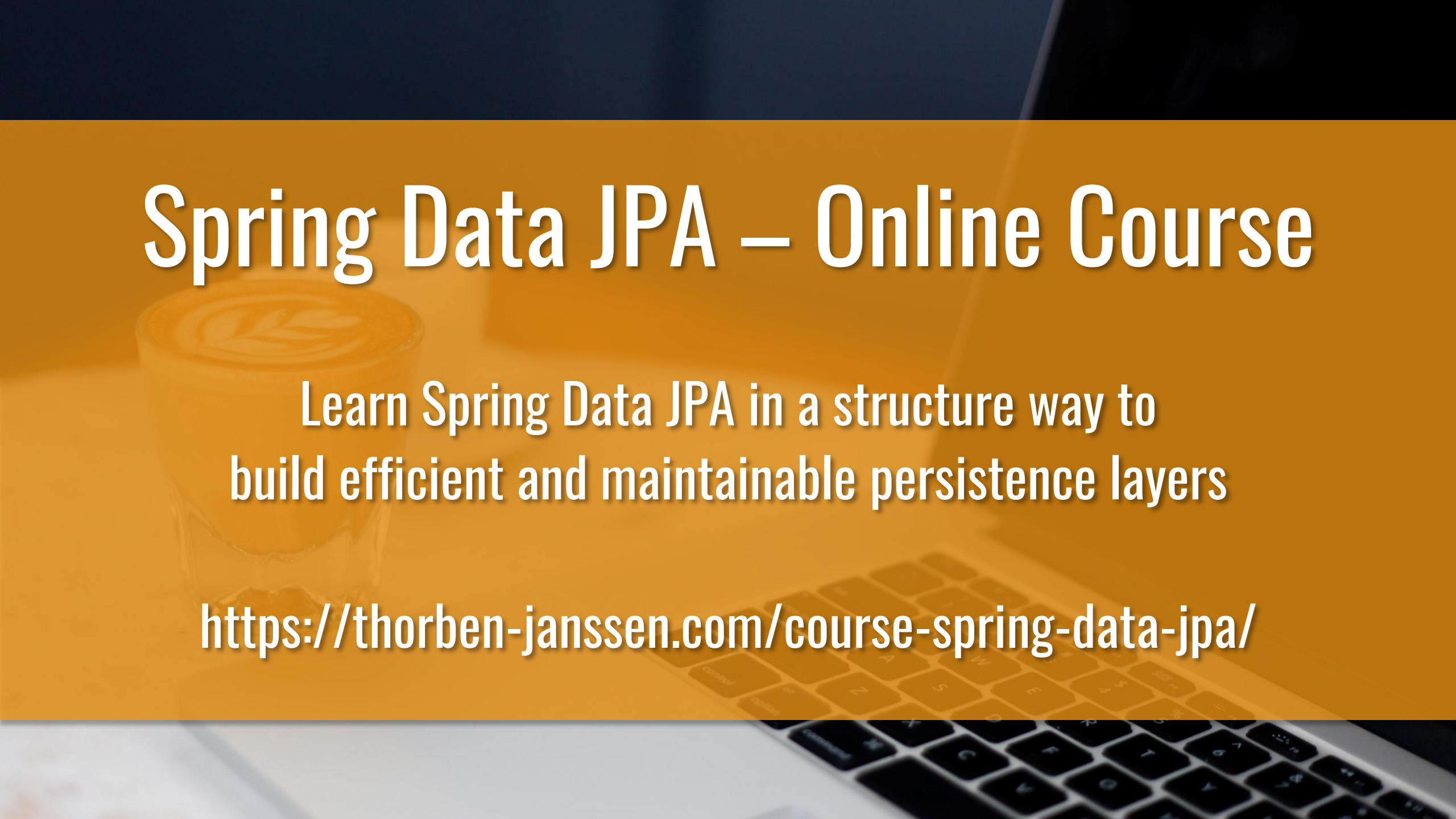
2nd Level Cache

- Session independent entity store
- Needs to be activated
  - application.properties
- Transparent usage
- PersistenceProvider doesn‘t need to provide it

Shared Cache Mode

ALL	all entities
NONE	no entities
ENABLE_SELECTIVE	requires activation
DISABLE_SELECTIVE	can be deactivated
UNSPECIFIED	use default settings

Code Samples


Spring Data JPA – Online Course
Learn Spring Data JPA in a structure way to
build efficient and maintainable persistence layers
https://thorben-janssen.com/course-spring-data-jpa/