Spring Workshop

Spring Data – Persistence and ORM made simple

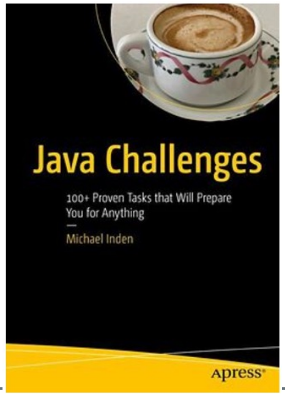
https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Michael Inden

Speaker Intro

- Michael Inden, Year of Birth 1971
- Diploma Computer Science, C.v.O. Uni Oldenburg
- ~8 ¼ Years **SSE** at Heidelberger Druckmaschinen AG in Kiel
- ~6 ¾ Years **TPL, SA** at IVU Traffic Technologies AG in Aachen
- ~4 ¼ Years **LSA / Trainer** at Zühlke Engineering AG in Zurich
- ~3 Years **TL / CTO** at Direct Mail Informatics / ASMIQ in Zurich
- Independent **Consultant, Conference Speaker and Trainer**
- **Since January 2022 Head of Development at Adcubum in Zurich**
- Author @ dpunkt.verlag and APress

E-Mail: michael_inden@hotmail.com
Blog: https://jaxenter.de/author/minden



Michael Inden
Java Challenge
Fit für das Job-Interview und die Praxis – mit mehr als 100 Aufgaben und Musterlösungen
dpunkt.verlag

5. Auflage
Michael Inden
Der Weg zum Java-Profi
Konzepte und Techniken für die professionelle Java-Entwicklung
dpunkt.verlag

https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

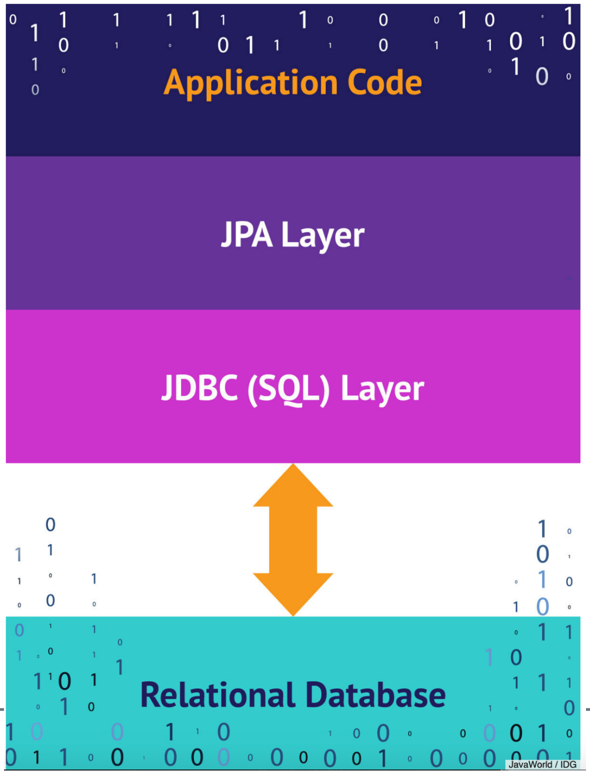
Agenda

Workshop Contents

PART 1: Introduction

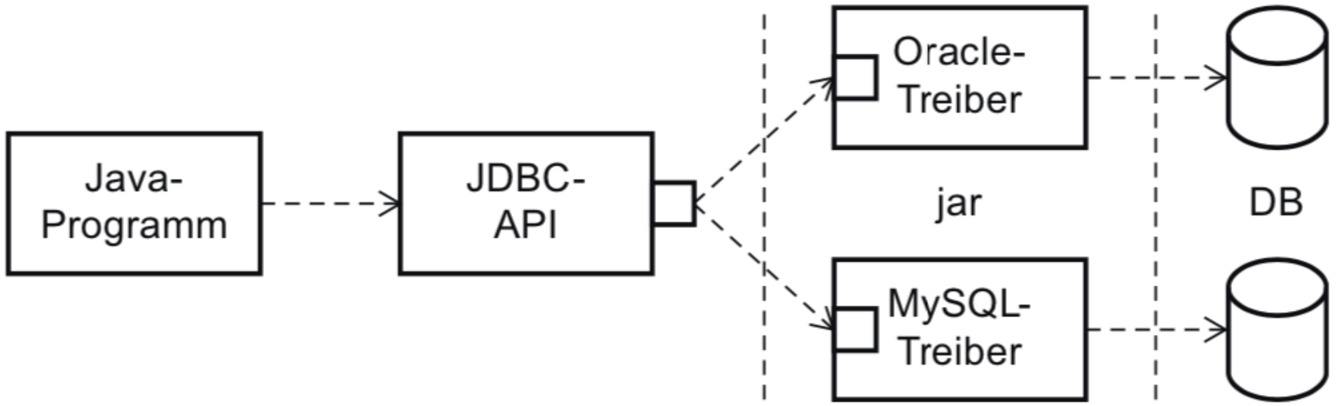
JPA Basics

- **Java Persistence API (JPA) is a collection of classes and interfaces for managing data in a database**
- **JPA "mediates" between application / domain model and databases (RDBMS)**



JDBC

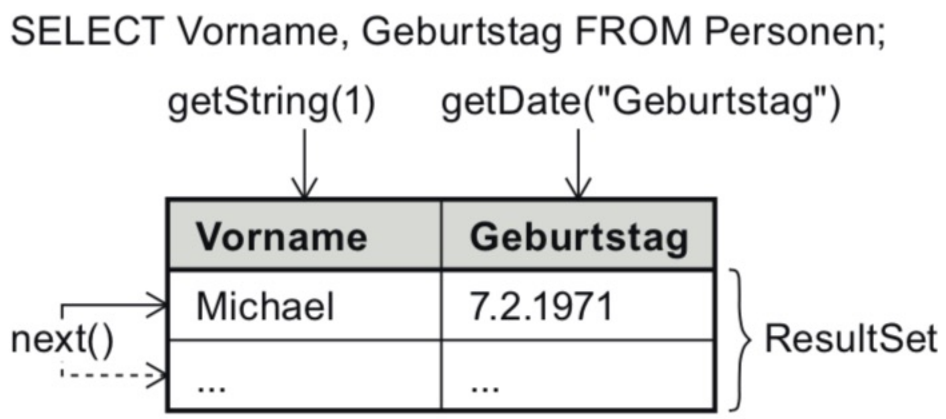
- **JDBC = Java Database Connectivity**
- **Defines a standard for accessing relational databases**



- **`javax.sql.DataSource` represents an abstract factory for connections to databases**

JDBC

- `java.sql.Connection` represents a connection to the database and generates statements and controls transactions etc.
- `java.sql.Statement` / `PreparedStatement` represents a statement to be executed
- `java.sql.ResultSet` represents results of queries



JPA vs JDBC

- **JPA significantly more high-level than JDBC**
  - No more JDBC/SQL statements needed by developers
  - Many JDBC/SQL commands are automatically generated by JPA
  - JPA is database independent, then adapts the SQL database specific
  - SQL like language (JPQL) for queries
  - Database is shining through, but much more abstraction
- **JPA is used for object/relational mapping (ORM)**
  - Application works with objects and the concept of entities
  - Entities are mapped to database tables
  - Commands and actions as methods and not SQL commands
  - Associations and inheritance can be realized without tricks

JPA Features

- **JPA automatically matches possible changes to the object/entity state with the database (at the end of the transaction).**
- **User modifies the object model and this is immediately mapped to the database, no additional SQL commands necessary**
- **Even sophisticated things like associations and inheritance are easily supported:**
  - Automatic loading of referenced objects possible
  - Automatic deletion of referenced objects possible
- **JPA manages the entities and caches them (1st level caching)**

JPA Good to know

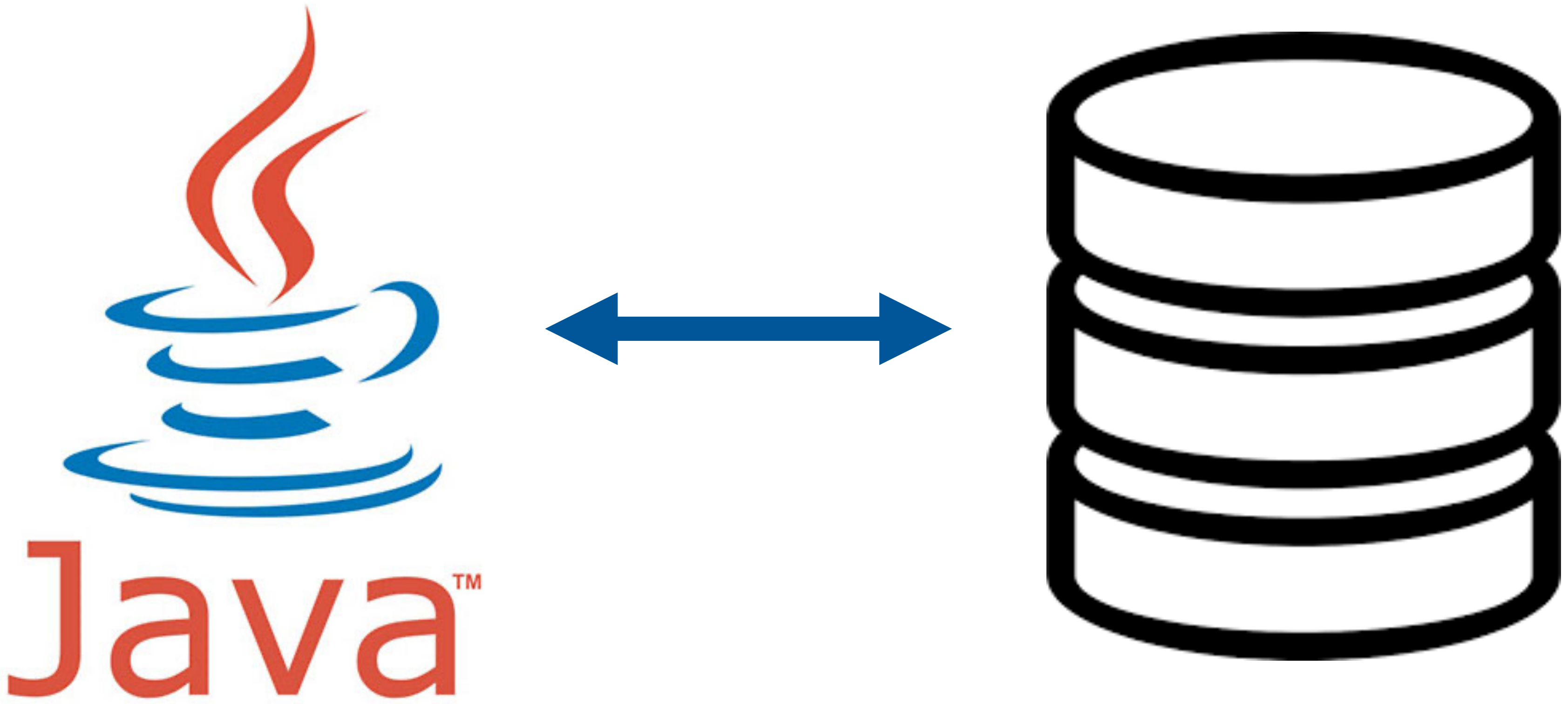
- **JPA is (only) a specification (not an implementation)**

- **JPA functionality is implemented by different providers**
  - Hibernate
  - EclipseLink / TopLink
  - ObjectDB

ORM Intro

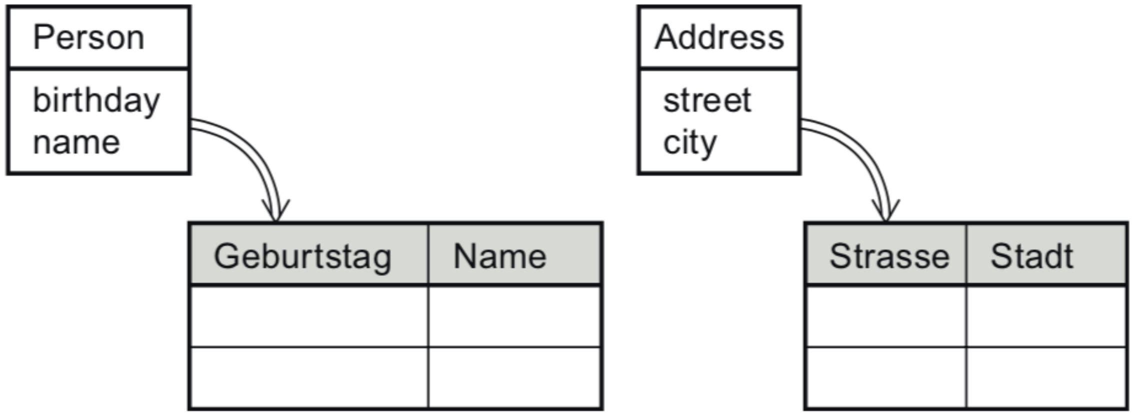
ORM = Object-Relational Mapping

- **ORM = Object-Relational Mapping**
- **Mapping from the object-oriented model to the database model**



ORM = Object-Relational Mapping

- **Map objects to tables in a database:**
  - Store objects in DB

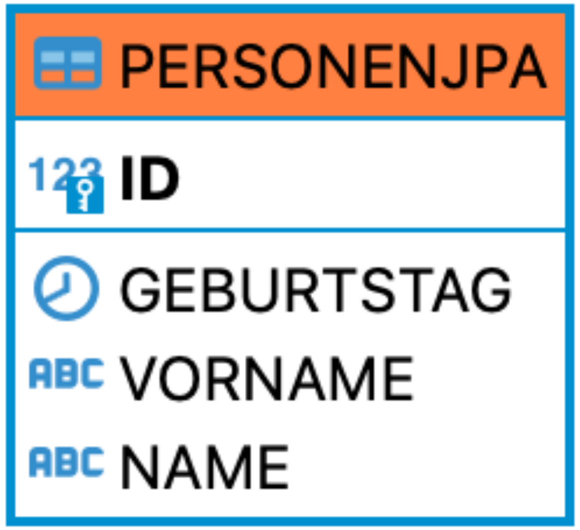


  - Reconstruct objects from DB

- **ORM can be programmed by yourself with some effort, far easier with JPA**
- **But it can be complex and challenging (especially for associations and inheritance).**

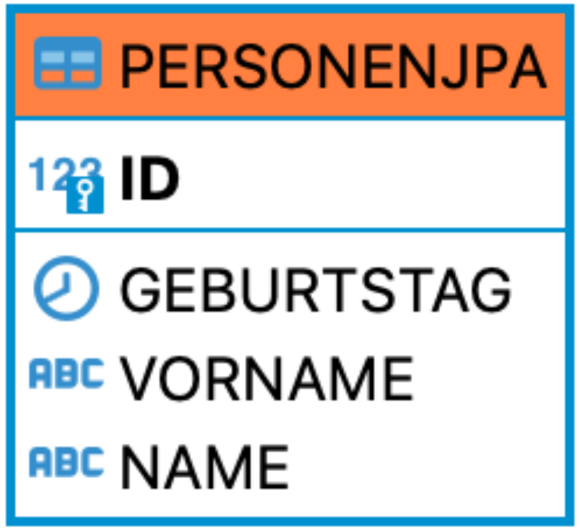
Example Entity: Variant Annotations on the methods

```
@Entity
@Table(name = "PersonenJPA")
public class Person implements Serializable
{
    private Long id;
    private String firstName;
    private String lastName;
    private LocalDate birthday;

    @Id
    @GeneratedValue
    public Long getId()  // read only
    {
        return id;
    }

    @Column(name = "Vorname")
    public String getFirstName()
    {
        return firstName;
    }

    ...
}
```



Example Entity: Variant Annotations on the attributes

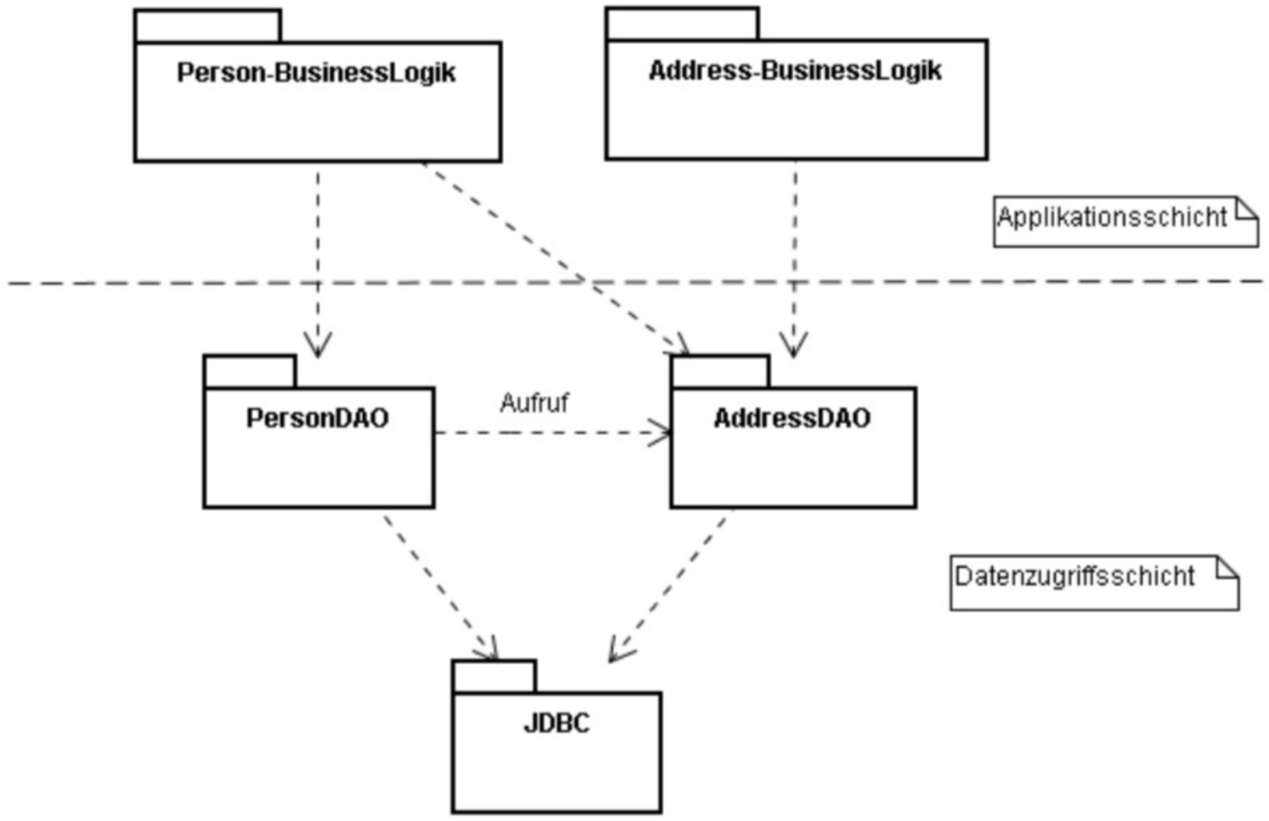
```
@Entity
@Table(name = "PersonenJPA")
public class Person implements Serializable
{
    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "Vorname")
    private String firstName;

    @Column(name = "Name")
    private String lastName;

    @Column(name = "Geburtstag")
    private LocalDate birthday;

    ...
}
```



DAOs and Repositories

DAO Pattern

- The Data Access Object (DAO) pattern is a structural pattern.
- It allows to isolate the application/business layer from the persistence layer using an abstract API.



DAO Pattern

- DAO hides all complexities associated with performing CRUD operations from the application.
- DAO leads to loose coupling: components of the application can more easily evolve separately from each other
- Closely related to the Repository Pattern (DAO is sometimes referred to as Repository).
- Ideally, all database access in the system is done through a DAO to achieve good encapsulation.

DAO Pattern

- **Each DAO instance is in charge of one entity, in particular the CRUD operations: Create, Read (by primary key), Update, and Delete (CRUD) the domain object.**

```
public interface IPersonDAO
{
    // CRUD funktionality
    List<Person> getAllPersons() throws DataAccessException;
    long addPerson(Person person) throws DataAccessException;
    void updateFromOther(long personId, Person otherPerson) throws DataAccessException;
    void removePerson(long personId) throws DataAccessException;
}
```

- **DAO can enable other actions, such as special queries**
- **DAO is not responsible for handling transactions, session or connections - these are handled outside DAO**

```java
private static void executeStatements(final EntityManager entityManager)
{
    // DAO creation
    final IPersonDAO dao = new PersonDAO(entityManager);

    // inserts and check result
    final Person michael = new Person("Micha-DAO", "Inden", new Date(71, 1, 7));
    final Person michael2 = new Person("Micha-DAO", "Inden", new Date(71, 1, 7));
    final Person werner = new Person("Werner-DAO", "Inden", new Date(40, 0, 31));

    final long michaelId = dao.createPerson(michael);
    final long michaelId2 = dao.createPerson(michael2);
    final long wernerId = dao.createPerson(werner);

    final List<Person> persons2 = dao.findAllPersons();
    persons2.forEach(System.out::println);

    // perform modifications and check result
    dao.deletePersonById(michaelId);
    werner.setFirstName("Dr. h.c. Werner");

    final List<Person> persons = dao.findAllPersons();
    persons.forEach(System.out::println);
}
```

More general DAO Pattern

- Rudimentary interface for a generic DAO
- save() is responsible for "create" and "update" respectively "update" happens by attribute changes

```java
import java.util.List;
import java.util.Optional;

public interface Dao<T>
{
    T save(T t);

    T get(long id);
    List<T> getAll();

    void delete(T t);
}
```

```java
import java.util.List;
import java.util.Optional;

public interface Dao<T>
{
    T save(T t);

    Optional<T> get(long id);
    List<T> getAll();

    void delete(T t);
}
```

More general DAO Pattern using JPA

- **Different types for keys**
- **Implementation and connection to `EntityManager`**

```
public final class GenericDAO<T, K>
{
    private final EntityManager entityManager;
    private final Class<T> clazz;

    GenericDAO(final EntityManager entityManager, final Class<T> clazz)
    {
        this.entityManager = entityManager;
        this.clazz = clazz;
    }

    // C -- CREATE
    public T save(final T newObject)
    {
        entityManager.persist(newObject);
        return newObject;
    }
```

More general DAO Pattern using JPA

```
// R -- READ
public T findById(final K id)
{
    return entityManager.find(clazz, id);
}

// R -- READ
public List<T> findAll()
{
   final TypedQuery<T> query = entityManager.createQuery("FROM " + clazz.getSimpleName(),
                                                         clazz);
   return query.getResultList();
}

// D -- DELETE
public void deleteById(final K id)
{
    final T objectInDb = findById(id);
    if (objectInDb != null)
    {
        entityManager.remove(objectInDb);
    }
}
```

That's already good. Wouldn't it be great if you could make it even shorter and more powerful?

Spring Data Repositories

- **Data Access Object (DAO) usually simplify data access enormously.**
- **Even better are the Spring Data Repositories, which already provide a collection of very useful methods out of the box**
- **You can easily define your own queries**

```
public interface IFooDAO extends JpaRepository<Foo, Long> {

    Foo findByName(String name);

}
```

- **More on that in some minutes ...**

Experiments with in-memory DB

H2 In-Memory-DB

- **H2 is one of the most popular in-memory databases. Spring Boot provides a very good integration for H2.**

- **H2 is a relational database management system written in Java. It can be embedded in Java applications or run in client-server mode.**

- **H2 supports a (substantial) subset of the SQL standard.**

- **H2 also provides a web console for database management.**

- **http://www.h2database.com/html/features.html**

Additional H2 Dependencies

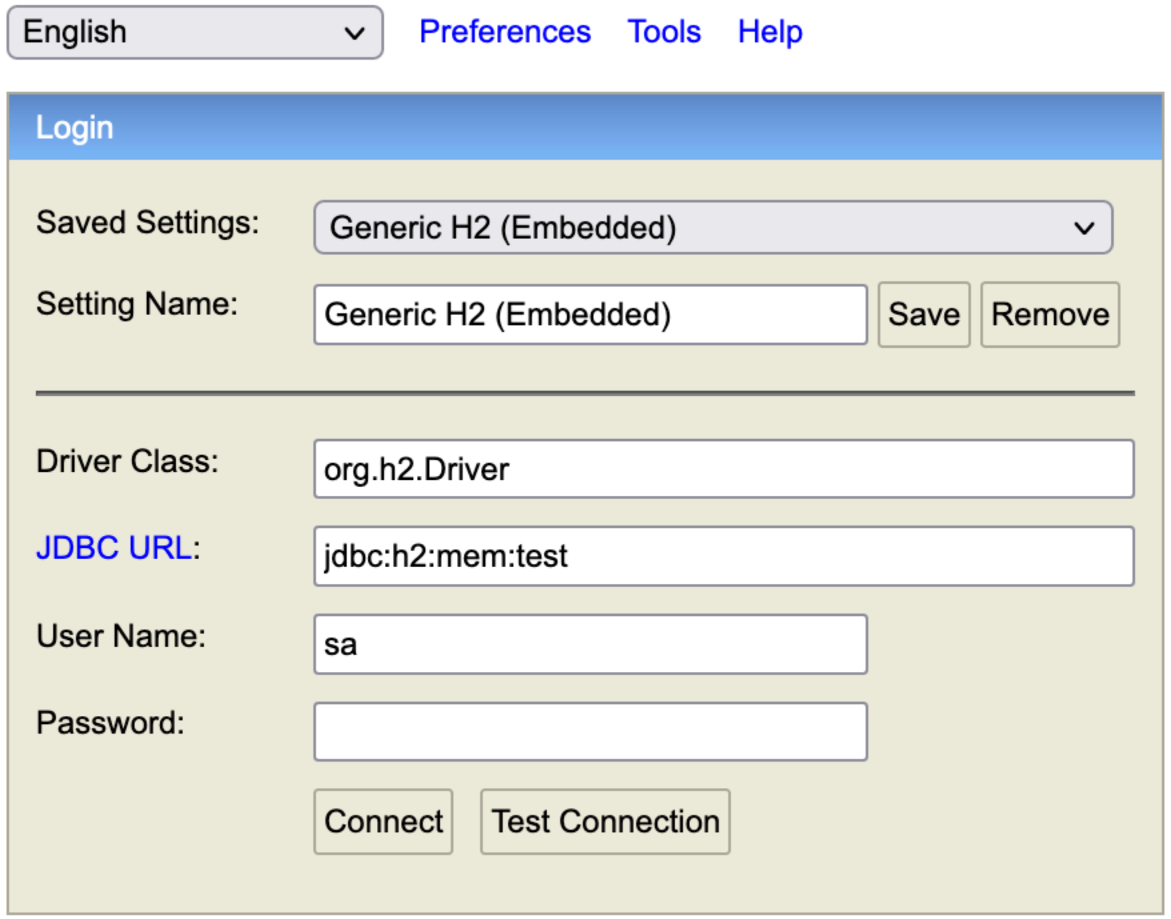
Maven

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.1.214</version>
</dependency>
```

Gradle

```
implementation group: 'com.h2database', name: 'h2', version: '2.1.214'
```

H2 Login

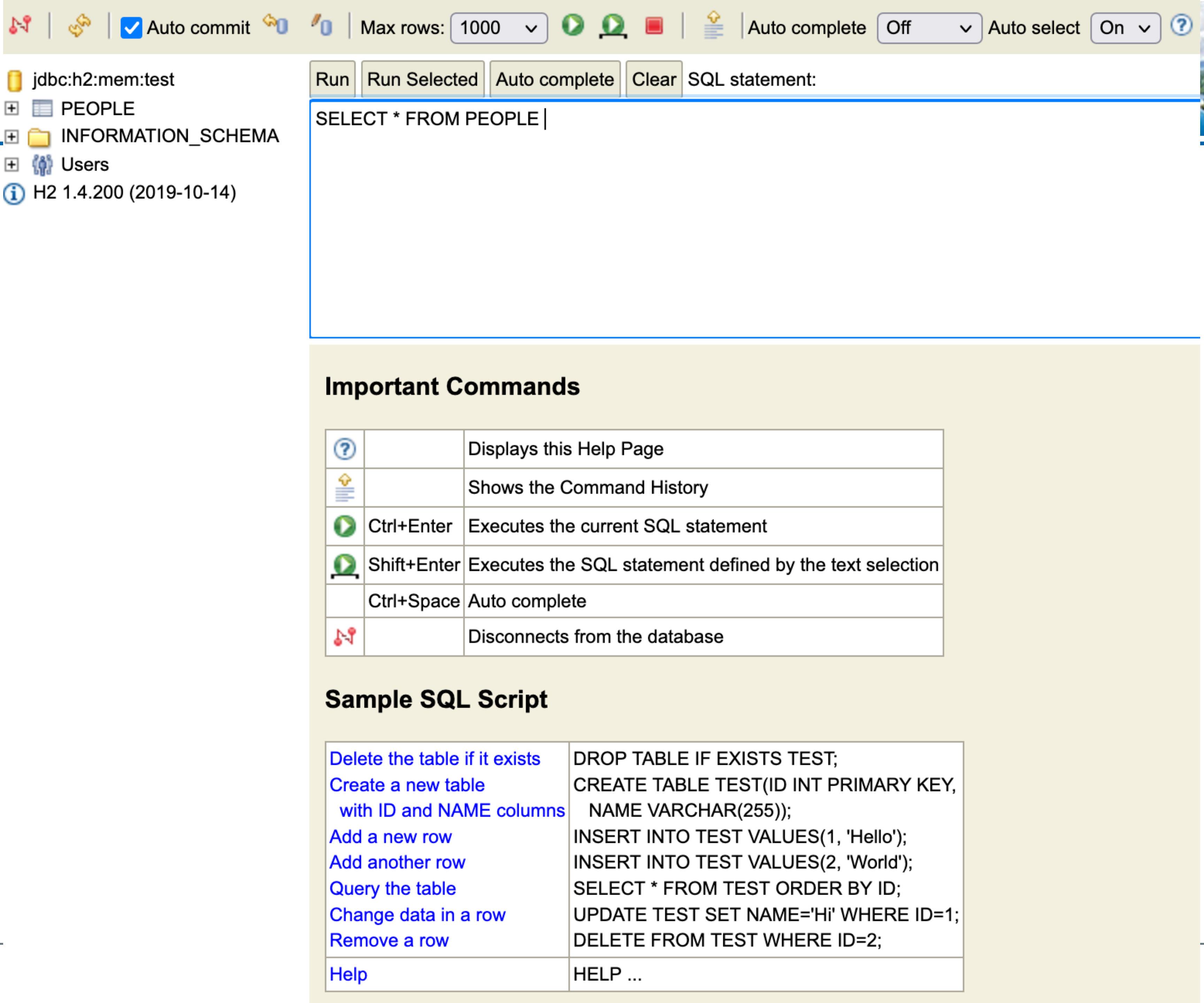


English | Preferences | Tools | Help

Login

Saved Settings:	Generic H2 (Embedded)
Setting Name:	Generic H2 (Embedded)
Driver Class:	org.h2.Driver
JDBC URL:	jdbc:h2:mem:test
User Name:	sa
Password:	

H2 Screen



Auto commit [✓] Max rows: 1000 Auto complete: Off Auto select: On

Run | Run Selected | Auto complete | Clear | SQL statement:

```
SELECT * FROM PEOPLE
```

jdbc:h2:mem:test
- PEOPLE
- INFORMATION_SCHEMA
- Users
- H2 1.4.200 (2019-10-14)

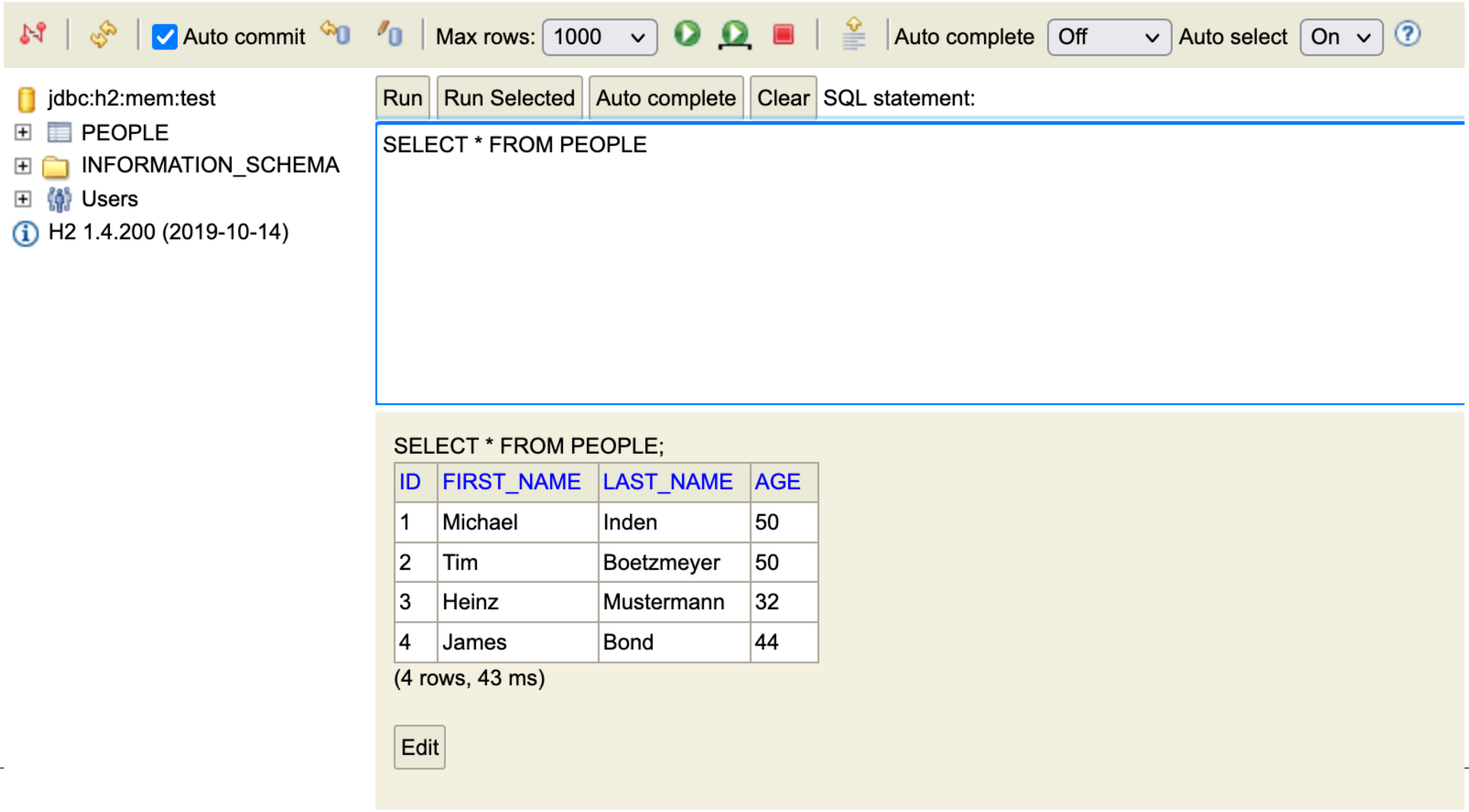
Important Commands

?		Displays this Help Page
		Shows the Command History
▶	Ctrl+Enter	Executes the current SQL statement
▶	Shift+Enter	Executes the SQL statement defined by the text selection
	Ctrl+Space	Auto complete
		Disconnects from the database

Sample SQL Script

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST(ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='Hi' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

H2 Query



```
SELECT * FROM PEOPLE
```

```
SELECT * FROM PEOPLE;
```

ID	FIRST_NAME	LAST_NAME	AGE
1	Michael	Inden	50
2	Tim	Boetzmeyer	50
3	Heinz	Mustermann	32
4	James	Bond	44

(4 rows, 43 ms)

H2 InMemoryDB interesting Links

- https://howtodoinjava.com/spring-boot2/h2-database-example/
- https://www.linkedin.com/pulse/unit-testing-using-h2-in-memory-db-raghunandan-gupta/
- https://phauer.com/2017/dont-use-in-memory-databases-tests-h2/

Part 2: Spring Data

- Introduction
- Spring Data JPA Basics
- Spring Data Repositories
- Spring Data Mongo DB

Introduction

Spring Data

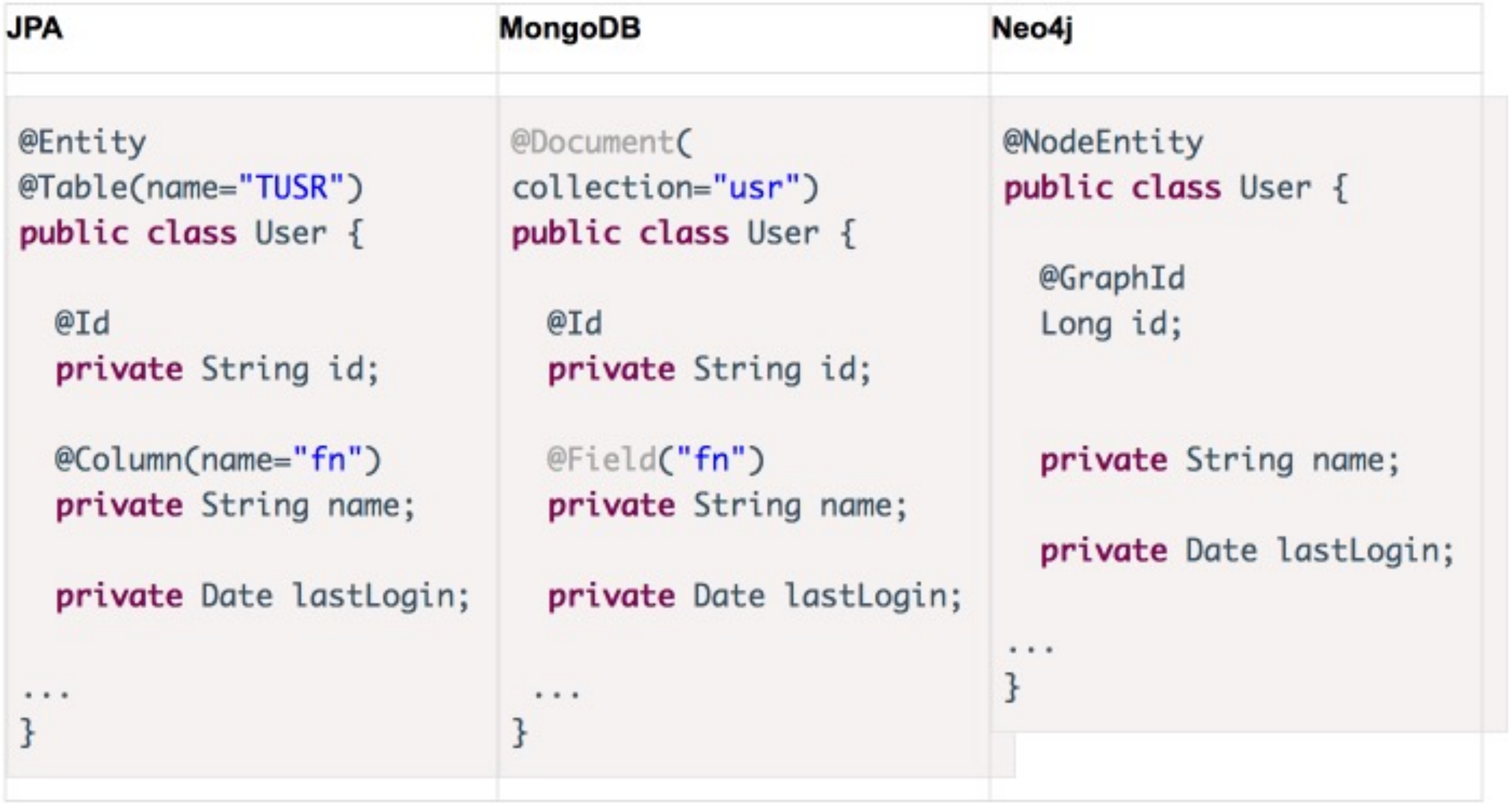
- provide a **familiar** and **consistent**, Spring-based programming model for **data access**
- makes it **easy** to use **relational** and **non-relational databases**, and **cloud-based data** services.
- **umbrella project** which contains **many subprojects** that are specific to a given database.

http://projects.spring.io/spring-data/

Spring Data Main Modules

- **Spring Data Commons** - Core Spring concepts underpinning every Spring Data project.
- **Spring Data JPA** - Makes it easy to implement JPA-based repositories.
- **Spring Data MongoDB** - Spring based, object-document support and repositories for MongoDB.
- …

Same and yet different



JPA

```java
@Entity
@Table(name="TUSR")
public class User {

  @Id
  private String id;

  @Column(name="fn")
  private String name;

  private Date lastLogin;

...
}
```

MongoDB

```java
@Document(
collection="usr")
public class User {

  @Id
  private String id;

  @Field("fn")
  private String name;

  private Date lastLogin;

 ...
}
```

Neo4j

```java
@NodeEntity
public class User {

  @GraphId
  Long id;


  private String name;

  private Date lastLogin;

...
}
```

Spring Data JPA Basics

Getting Started — Maven Dependencies

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.5.6</version>
</parent>


<dependencies>
    <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
    </dependency>
...
```

Getting Started — Gradle Dependencies

```
plugins {
  id "org.springframework.boot" version "2.5.6"
}

apply plugin: 'java'
apply plugin: 'eclipse'


repositories {
    mavenCentral()
}

sourceCompatibility = 11
targetCompatibility = 11

dependencies {

    implementation 'org.springframework.boot:spring-boot-starter-data-jpa:2.5.6'
    testImplementation 'org.springframework.boot:spring-boot-starter-test:2.5.6'
```

First Spring Boot Application Example

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class MyApp {

   public static void main(String[] args) {
      SpringApplication.run(MyApp.class, args);
   }
}
```

First Entity Example

```
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class SimpleEmployee
{
    @Id
    @GeneratedValue
    private Long id;
    private String firstName, lastName, description;

    private SimpleEmployee()
    {
    }

    public SimpleEmployee(String firstName, String lastName, String description)
    {
        this.firstName = firstName;
        this.lastName = lastName;
        this.description = description;
    }
...
```

First Repository Example

- Database queries follow the **DAO pattern**
- These are described by so-called **repositories**

- In Spring these are **simple interfaces (POJI) => declarative programming**

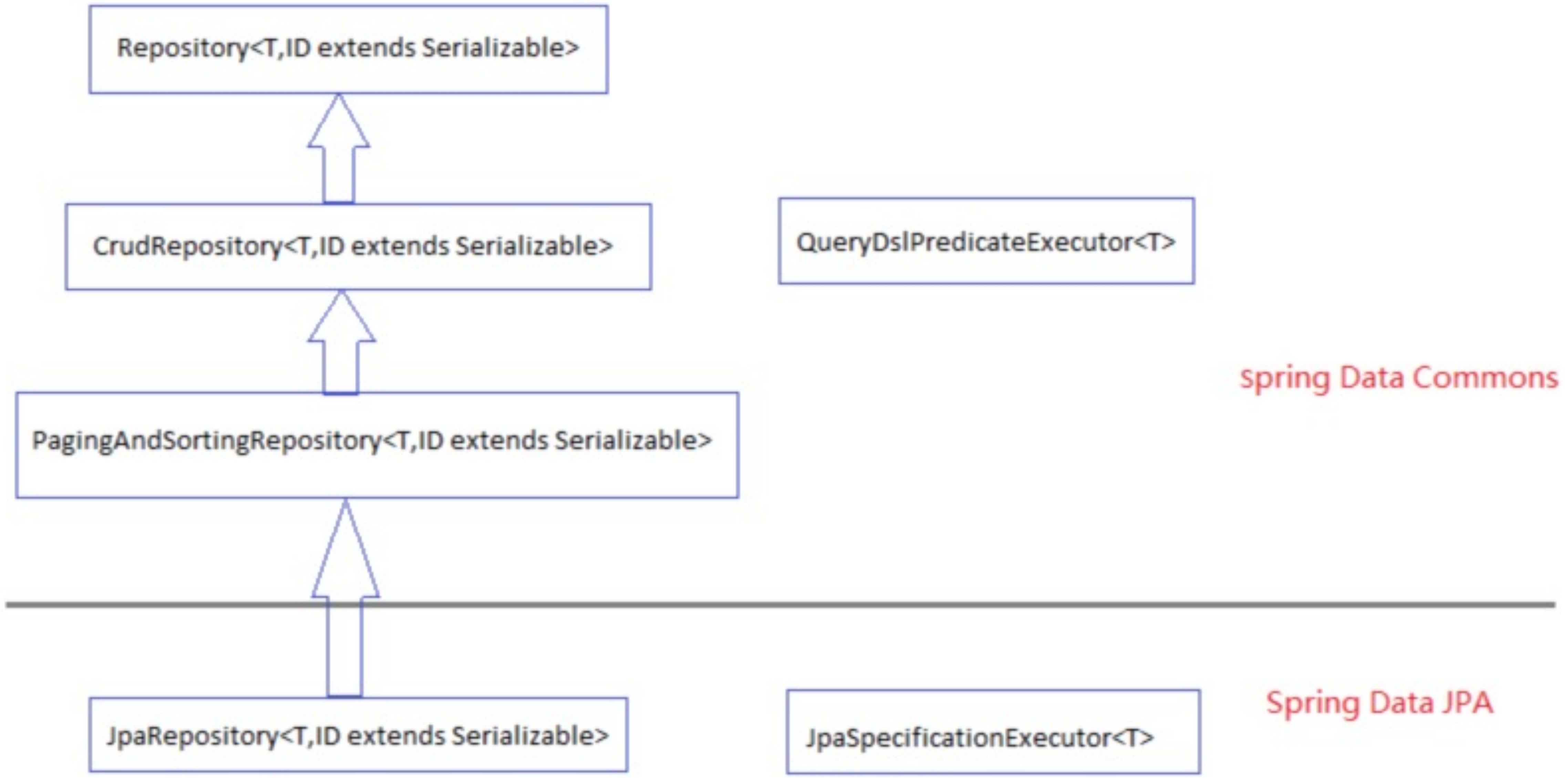
```
import java.util.List;

import org.springframework.data.repository.CrudRepository;

public interface SimpleEmployeeRepository extends CrudRepository<SimpleEmployee, Long>
{
    SimpleEmployee findByFirstName(String firstName);

    List<SimpleEmployee> findByLastName(String lastName);
}
```

- **Database accesses are generated automatically based on `findXyz()`**

Basis Spring CRUD Repository — Default functionality

```
public interface CrudRepository<T, ID extends Serializable> extends Repository<T, ID> {

  <S extends T> S save(S entity);

  Optional<T> findById(ID primaryKey);

  Iterable<T> findAll();

  long count();

  void delete(T entity);

  boolean existsById(ID primaryKey);

  // …
}
```

Basis Spring Repositories


Repository<T,ID extends Serializable>
CrudRepository<T,ID extends Serializable>
QueryDslPredicateExecutor<T>
Spring Data Commons
PagingAndSortingRepository<T,ID extends Serializable>
Spring Data JPA
JpaRepository<T,ID extends Serializable>
JpaSpecificationExecutor<T>

First Example

Let's just start …

```
@SpringBootApplication
public class MyApp {

   public static void main(String[] args) {
      SpringApplication.run(MyApp.class, args);
   }
}
```

```
***************************
APPLICATION FAILED TO START
***************************

Description:

Cannot determine embedded database driver class for database type NONE
```

How do we include a DB?

H2 and Spring Boot

- Configuring the H2 database with Spring Boot is very simple: just add the H2 dependency to the POM:
- Spring Boot automatically creates the database, sets up all the database JDBC objects, and configures Hibernate in a create-drop mode by default.
- When Hibernate starts, it scans the JPA annotated classes and automatically generates and executes the SQL code required to create the database tables.

RECAP: Additional H2 Dependencies

Maven

```xml
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>2.1.214</version>
</dependency>
```

Gradle

```groovy
implementation group: 'com.h2database', name: 'h2', version: '2.1.214'
```

Launch application

Maven

```
mvn clean package
mvn spring-boot:run
```

```
<plugin>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-maven-plugin</artifactId>
</plugin>
```

Gradle

```
gradle clean assemble
gradle bootRun
```

```
plugins { id 'org.springframework.boot' version '2.6.0' }
```

Command line

```
mvn clean package spring-boot:repackage
java -jar target/ex21-spring-boot-person-datajpa-app-1.0.0.jar
```

Launch application

Maven

```
mvn clean package
mvn spring-boot:run
```

Gradle

```
gradle clean assemble
gradle bootRun
```

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::                (v2.5.6)
```

Command line

```
mvn clean package spring-boot:repackage
java -jar target/ex21-spring-boot-person-datajpa-app-1.0.0.jar
```

How do we work with the DB?

Populate database — CommandLineRunner (or scripts)

```
@SpringBootApplication
public class Application implements CommandLineRunner
{
   @Autowired
   private SimpleEmployeeRepository repository;

   public static void main(String[] args)
   {
      SpringApplication.run(Application.class, args);
   }

   public void run(String... args) throws Exception
   {
      …
   }
}
```

Populate database — CommandLineRunner

```
public void run(String... args) throws Exception
{
   Employee emp1 = new Employee("Michael", "Inden", "Team Lead");
   Employee emp2 = new Employee("Karthi", "Bollu Ganesh", "Lead Engineer");
   Employee emp3 = new Employee("Marcello", "Fluri", "Senior SW Engineer");

   System.out.println("Employees: " + repository.count());
   repository.save(emp1);
   repository.save(emp2);
   repository.save(emp3);
   System.out.println("Employees: " + repository.count());
   System.out.println("Employees: " + repository.findAll());

   // Find + Delete
   repository.delete(repository.findByFirstName("Marcello"));
   System.out.println("Employees: " + repository.count());
   System.out.println("Employees: " + repository.findAll());
}
```

Populate database — CommandLineRunner

```
Employees: 0
Employees: 3
Employees: [SimpleEmployee [id=6, firstName=Michael, lastName=Inden, description=Team Lead],
SimpleEmployee [id=7, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer],
SimpleEmployee [id=8, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer]]
Employees: 2
Employees: [SimpleEmployee [id=6, firstName=Michael, lastName=Inden, description=Team Lead],
SimpleEmployee [id=7, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer]]
```

Spring Data Repositories

Spring Data Repositories: Query variants via method names

```
public interface MovieRepository extends JpaRepository<Movie, Long>
{
    List<Movie> findByTitleIgnoringCase(String title);
}
```

- **findBy, readBy, getBy, countBy, queryBy**
- **GreaterThan, LessThan, Between**
- **Like, In**

- **Sorting: OrderBy…Asc / Desc**
- **Uniqueness: Distinct**
- **Restrictions / Paging: Top / First, e.g. Top10**

Spring Data Repositories

Possible variants

- **`And, Or`**
  - `findByLastnameAndFirstname() / findByLastnameOrFirstname()`
  - `… where x.lastname = ?1 and (or) x.firstname = ?2`

- **`Is, Equals`**

  `findByFirstnameIs() / findByFirstnameEquals() / findByFirstname()`
  `… where x.firstname = ?1`

- **`Between`**
  - `findByStartDateBetween()`
    `… where x.startDate between ?1 and ?2`

- **`LessThan, GreaterThan`**
  - `findByAgeLessThan() / findByAgeGreaterThan()`
    `… where x.age < ?1 / … where x.age > ?1`

https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods

Spring Data Repositories

Possible variants

- `After, Before`
- `IsNull, IsNotNull, NotNull`
- `Like / NotLike`
- `Containing`
- `OrderBy`
- `True / False`
- `In / NotIn`
- `Not`
- `IgnoreCase`
- `Asc / Desc`

- **Limit the result size of a query**
  - `findFirst10ByLastnameAsc`

Keywords

Keyword	Sample	JPQL snippet
`And`	`findByLastnameAndFirstname`	`… where x.lastname = ?1 and x.firstname = ?2`
`Or`	`findByLastnameOrFirstname`	`… where x.lastname = ?1 or x.firstname = ?2`
`Is,Equals`	`findByFirstname,findByFirstnameIs,findByFirstnameEquals`	`… where x.firstname = 1?`
`Between`	`findByStartDateBetween`	`… where x.startDate between 1? and ?2`
`LessThan`	`findByAgeLessThan`	`… where x.age < ?1`
`LessThanEqual`	`findByAgeLessThanEqual`	`… where x.age <= ?1`
`GreaterThan`	`findByAgeGreaterThan`	`… where x.age > ?1`
`GreaterThanEqual`	`findByAgeGreaterThanEqual`	`… where x.age >= ?1`
`After`	`findByStartDateAfter`	`… where x.startDate > ?1`
`Before`	`findByStartDateBefore`	`… where x.startDate < ?1`
`IsNull`	`findByAgeIsNull`	`… where x.age is null`
`IsNotNull,NotNull`	`findByAge(Is)NotNull`	`… where x.age not null`
`Like`	`findByFirstnameLike`	`… where x.firstname like ?1`
`NotLike`	`findByFirstnameNotLike`	`… where x.firstname not like ?1`
`StartingWith`	`findByFirstnameStartingWith`	`… where x.firstname like ?1` (parameter bound with appended `%`)
`EndingWith`	`findByFirstnameEndingWith`	`… where x.firstname like ?1` (parameter bound with prepended `%`)
`Containing`	`findByFirstnameContaining`	`… where x.firstname like ?1` (parameter bound wrapped in `%`)
`OrderBy`	`findByAgeOrderByLastnameDesc`	`… where x.age = ?1 order by x.lastname desc`
`Not`	`findByLastnameNot`	`… where x.lastname <> ?1`
`In`	`findByAgeIn(Collection<Age> ages)`	`… where x.age in ?1`
`NotIn`	`findByAgeNotIn(Collection<Age> age)`	`… where x.age not in ?1`

Repository Example

```
public interface EmployeeRepository extends CrudRepository<Employee, Long>
{
   Employee findByFirstName(String firstName);
   List<Employee> findByLastName(String lastName);

   List<Employee> findByAgeGreaterThan(int age);
   List<Employee> findByAgeBetween(int lower, int upper);
   int countByAgeBetween(int lower, int upper);

   List<Employee> findTop3ByAgeLessThan(int maxAge);
   List<Employee> findByAgeLessThanOrderByFirstNameAsc(int maxAge);

   @Query("SELECT emp FROM Employee emp WHERE emp.firstName LIKE %?1%")
   List<Employee> getFirstNameLike(String firstName);

   List<Employee> findByFirstNameOrLastName(String firstName, String lastName);

   List<Employee> findByLastNameInAndAgeBetween(Collection<String> names,
                                                int lower, int upper);
}
```

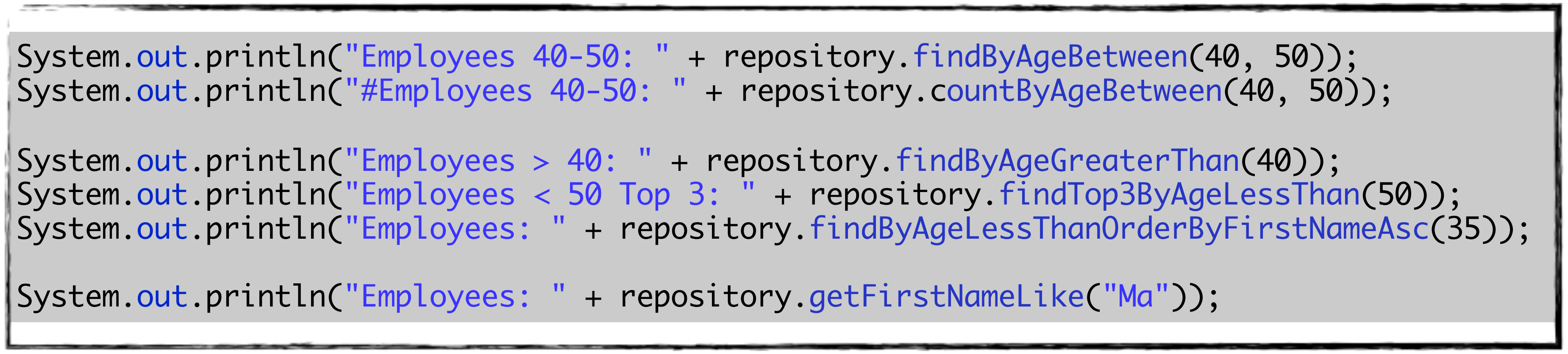
Repository Example

```
Employee emp1 = new Employee("Michael", "Inden", "Team Lead", 47);
Employee emp2 = new Employee("Karthi", "Bollu Ganesh", "Lead Engineer", 33);
Employee emp3 = new Employee("Marcello", "Fluri", "Senior SW Engineer", 52);
Employee emp4 = new Employee("Marco", "Sonderegger", "SW Engineer", 30);
Employee emp5 = new Employee("Numa", "Trezzini", "SW Engineer", 30);
Employee emp6 = new Employee("Martin", "Dorta", "Senior SW Engineer", 50);

employeeRepository.save(emp1);
employeeRepository.save(emp2);
employeeRepository.save(emp3);
employeeRepository.save(emp4);
employeeRepository.save(emp5);
employeeRepository.save(emp6);
```

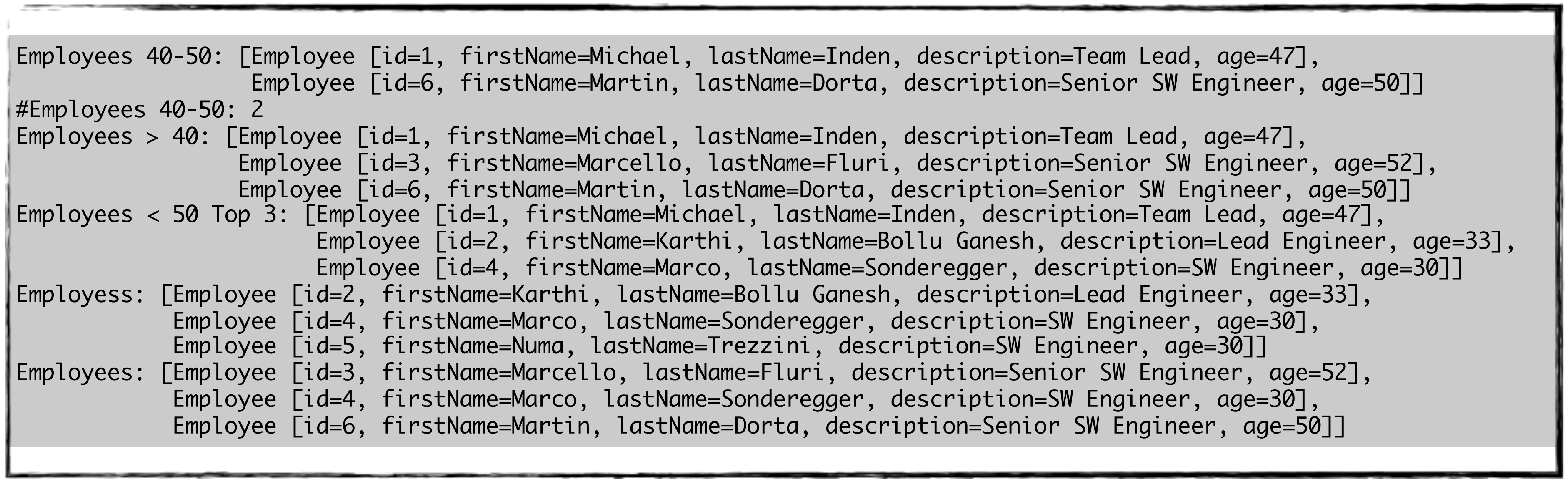
Repository Example

```
System.out.println("Employees 40-50: " + employeeRepository.findByAgeBetween(40, 50));
System.out.println("#Employees 40-50: " + employeeRepository.countByAgeBetween(40, 50));

System.out.println("Employees > 40: " + employeeRepository.findByAgeGreaterThan(40));
System.out.println("Employees < 50 Top 3: " + employeeRepository.findTop3ByAgeLessThan(50));
System.out.println("Employess: " + employeeRepository.findByAgeLessThanOrderByFirstNameAsc(35));
System.out.println("Employees: " + employeeRepository.getFirstNameLike("Ma"));
System.out.println("Employees: " + employeeRepository.findByFirstNameOrLastName("Michael",
                                                                                "Fluri"));
```

Repository Example



```java
System.out.println("Employees 40-50: " + repository.findByAgeBetween(40, 50));
System.out.println("#Employees 40-50: " + repository.countByAgeBetween(40, 50));

System.out.println("Employees > 40: " + repository.findByAgeGreaterThan(40));
System.out.println("Employees < 50 Top 3: " + repository.findTop3ByAgeLessThan(50));
System.out.println("Employees: " + repository.findByAgeLessThanOrderByFirstNameAsc(35));

System.out.println("Employees: " + repository.getFirstNameLike("Ma"));
```



```
Employees 40-50: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                  Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
#Employees 40-50: 2
Employees > 40: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                 Employee [id=3, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
                 Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees < 50 Top 3: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                       Employee [id=2, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
                       Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30]]
Employess: [Employee [id=2, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
            Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
            Employee [id=5, firstName=Numa, lastName=Trezzini, description=SW Engineer, age=30]]
Employees: [Employee [id=3, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
            Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
            Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
```

DEMO

«spring-data-slides-examples»

Spring Data Repositories

- **Spring Data Repositories can be used to define not only custom queries based on method name, but also special SQL-like ones:**

  - Using JPQL

```
@Query("SELECT u FROM User u WHERE u.status = 1")
Collection<User> findAllActiveUsers();
```

```
~~@Query("SELECT u FROM User u WHERE u.status = ?1 and u.name = ?2")~~
User findUserByStatusAndName(Integer status, String name);
```

  - Using Native Queries

```
@Query(
  value = "SELECT * FROM USERS u WHERE u.status = 1",
  nativeQuery = true)
Collection<User> findAllActiveUsersNative();
```

Spring Data Repositories

- **Other possibilities**

```
@Query(value = “SELECT u FROM User u WHERE u.name IN :names”)
List<User> findUserByNameList(@Param(“names”) Collection<String> names);
```

```
@Modifying
@Query(“update User u set u.status = :status where u.name = :name”)
int updateUserSetStatusForName(@Param(“status”) Integer status,
  @Param(“name”) String name);
```

Exercises 21 – 22

https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Spring Data MongoDB Example

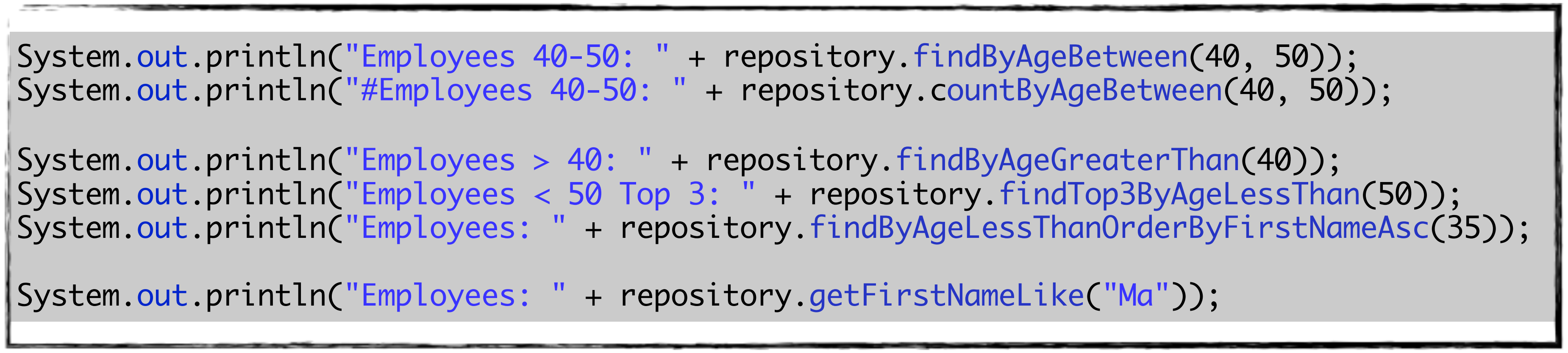
Repository Example

```
import org.springframework.data.annotation.Id;

import org.springframework.data.mongodb.core.mapping.Document;

@Document
public class Employee
{
   @Id
   private String id;


      …
}
```

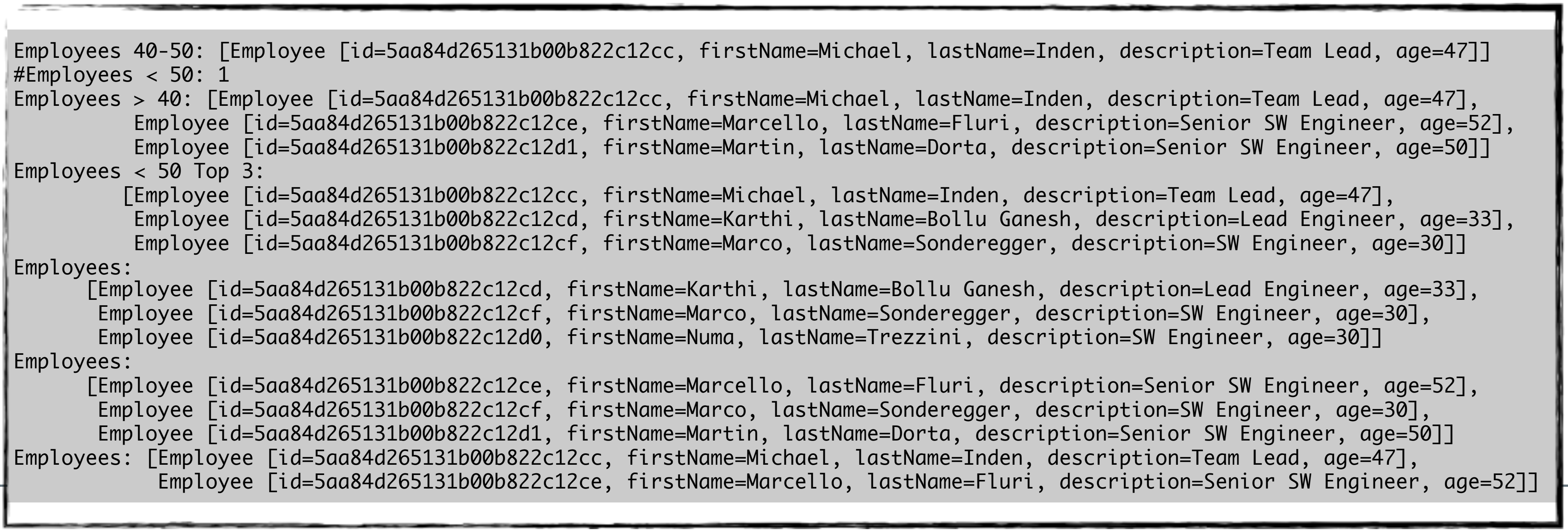
Repository Example

```
public interface EmployeeRepository extends MongoRepository<Employee, String>
{
   Employee findByFirstName(String firstName);
   List<Employee> findByLastName(String lastName);

   List<Employee> findByAgeGreaterThan(int age);
   List<Employee> findByAgeBetween(int lower, int upper);
   int countByAgeBetween(int lower, int upper);

   List<Employee> findTop3ByAgeLessThan(int maxAge);
   List<Employee> findByAgeLessThanOrderByFirstNameAsc(int maxAge);

   List<Employee> getByFirstNameLike(String firstName);

   List<Employee> findByFirstNameOrLastName(String firstName, String lastName);

   List<Employee> findByLastNameInAndAgeBetween(Collection<String> names,
                                                int lower, int upper);
}
```

Repository Example



```
System.out.println("Employees 40-50: " + repository.findByAgeBetween(40, 50));
System.out.println("#Employees 40-50: " + repository.countByAgeBetween(40, 50));

System.out.println("Employees > 40: " + repository.findByAgeGreaterThan(40));
System.out.println("Employees < 50 Top 3: " + repository.findTop3ByAgeLessThan(50));
System.out.println("Employees: " + repository.findByAgeLessThanOrderByFirstNameAsc(35));

System.out.println("Employees: " + repository.getFirstNameLike("Ma"));
```



```
Employees 40-50: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47]]
#Employees < 50: 1
Employees > 40: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
          Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
          Employee [id=5aa84d265131b00b822c12d1, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees < 50 Top 3:
         [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
          Employee [id=5aa84d265131b00b822c12cd, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
          Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30]]
Employees:
      [Employee [id=5aa84d265131b00b822c12cd, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
       Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
       Employee [id=5aa84d265131b00b822c12d0, firstName=Numa, lastName=Trezzini, description=SW Engineer, age=30]]
Employees:
      [Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
       Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
       Employee [id=5aa84d265131b00b822c12d1, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
            Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52]]
```

Repository Example

MongoDB: BETWEEN: lower < x < upper

```
Employees 40-50: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47]]
#Employees 40-50: 1
Employees > 40: [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
          Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
          Employee [id=5aa84d265131b00b822c12d1, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees < 50 Top 3:
         [Employee [id=5aa84d265131b00b822c12cc, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
          Employee [id=5aa84d265131b00b822c12cd, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
          Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30]]
Employees:
      [Employee [id=5aa84d265131b00b822c12cd, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
       Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
       Employee [id=5aa84d265131b00b822c12d0, firstName=Numa, lastName=Trezzini, description=SW Engineer, age=30]]
Employees:
      [Employee [id=5aa84d265131b00b822c12ce, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
       Employee [id=5aa84d265131b00b822c12cf, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
       Employee [id=5aa84d265131b00b822c12d1, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
```

JPA: BETWEEN: lower <= x <= upper

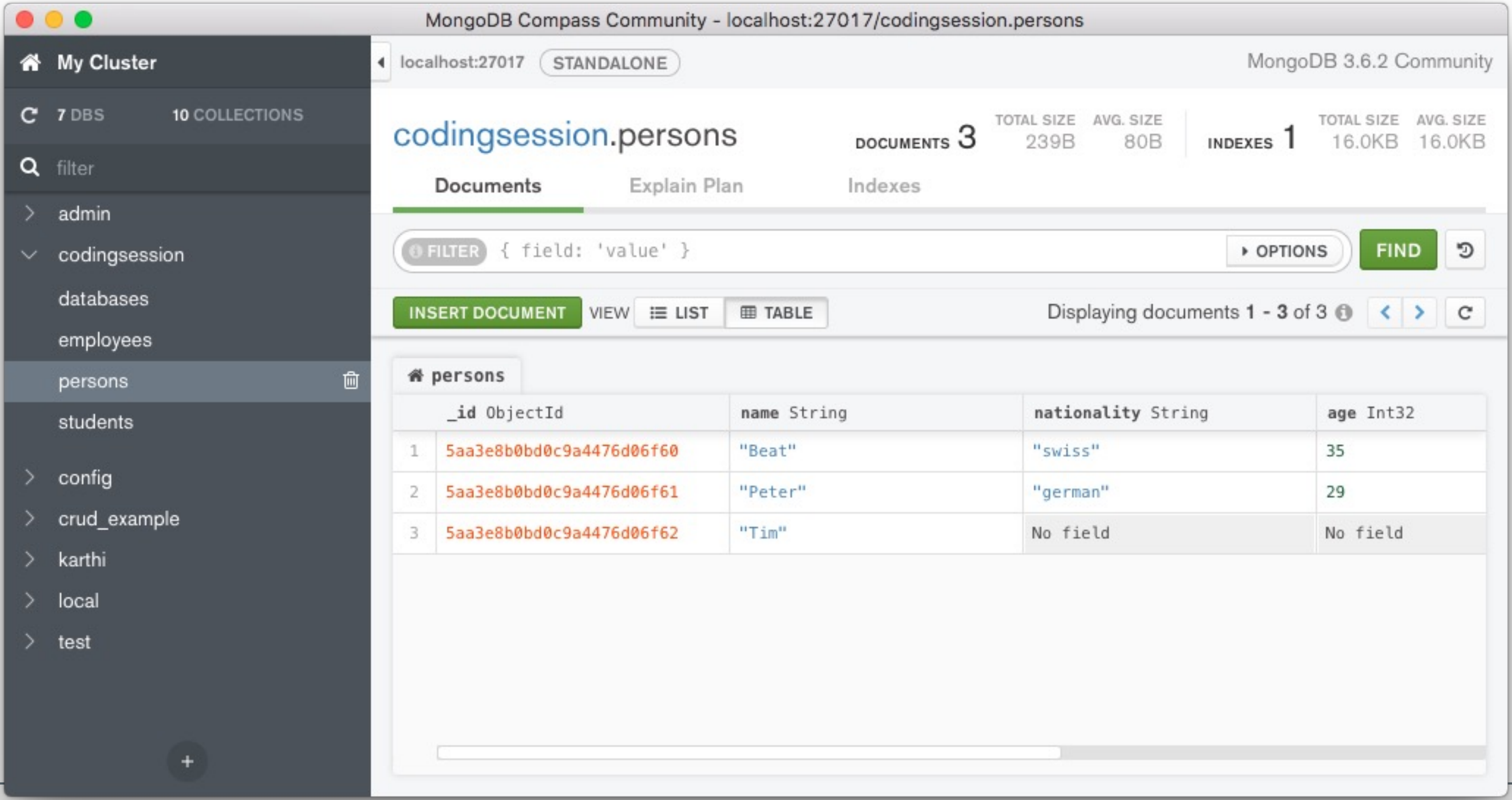
```
Employees 40-50: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                  Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
#Employees 40-50: 2
Employees > 40: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                 Employee [id=3, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
                 Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
Employees < 50 Top 3: [Employee [id=1, firstName=Michael, lastName=Inden, description=Team Lead, age=47],
                       Employee [id=2, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
                       Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30]]
Employees: [Employee [id=2, firstName=Karthi, lastName=Bollu Ganesh, description=Lead Engineer, age=33],
            Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
            Employee [id=5, firstName=Numa, lastName=Trezzini, description=SW Engineer, age=30]]
Employees: [Employee [id=3, firstName=Marcello, lastName=Fluri, description=Senior SW Engineer, age=52],
            Employee [id=4, firstName=Marco, lastName=Sonderegger, description=SW Engineer, age=30],
            Employee [id=6, firstName=Martin, lastName=Dorta, description=Senior SW Engineer, age=50]]
```

Repository Example

Names-Postfix	Operation als JSON
GreaterThan	`{ "age" : { "$gt" : <value> } }`
LessThan	`{ "age" : { "$lt" : <value> } }`
Between	`{ "age" : { "$gt" : from, "$lt" : to } }`
IsNotNull, NotNull	`{ "age" : { "$ne" : null } }`
IsNull, Null	`{ "age" : null }`
-/-	`{ "age" : <value> }`
Not	`{ "age" : { "$ne" : <value> } }`

MongoDB Compass

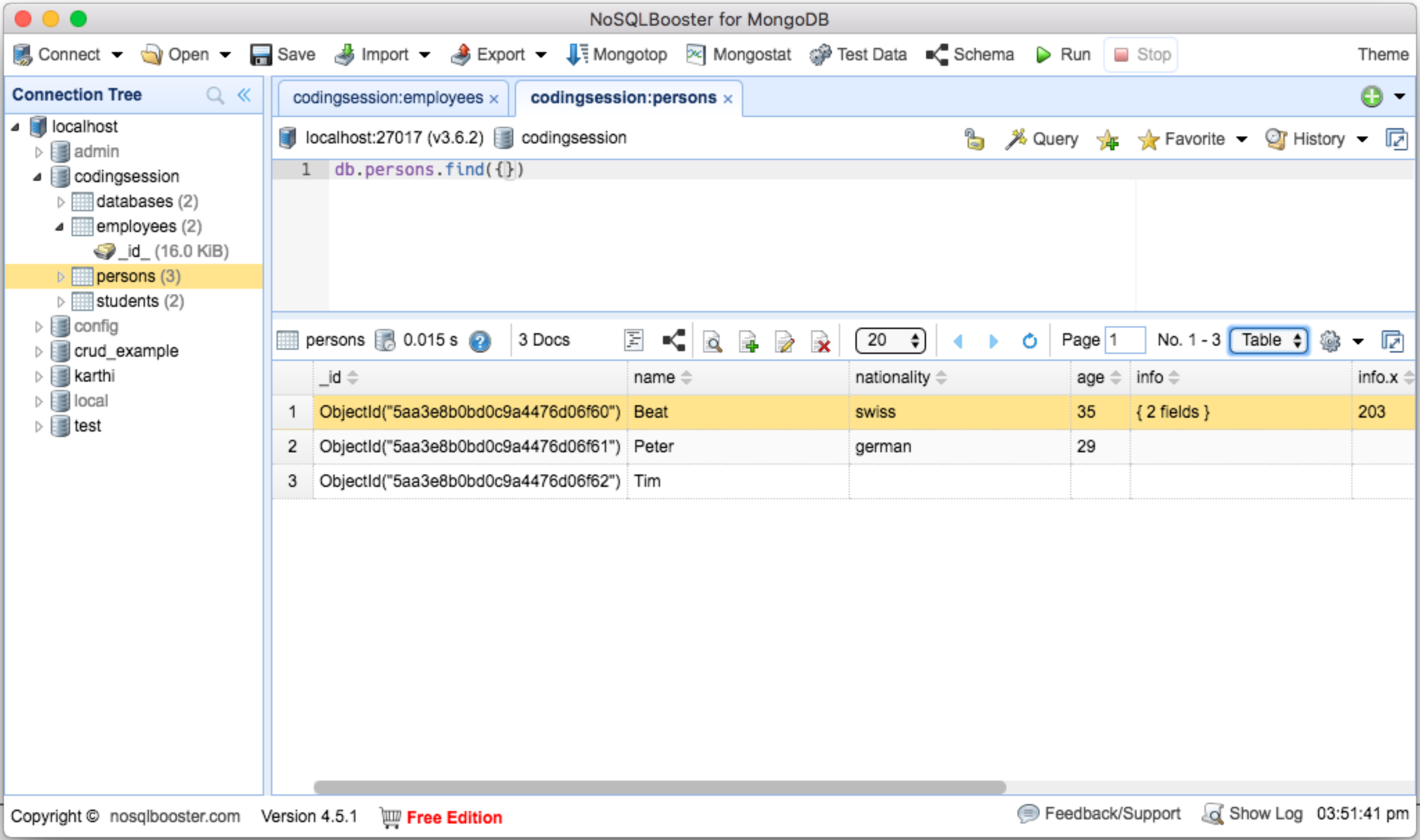
https://docs.mongodb.com/compass/master/install/



	_id ObjectId	name String	nationality String	age Int32
1	5aa3e8b0bd0c9a4476d06f60	"Beat"	"swiss"	35
2	5aa3e8b0bd0c9a4476d06f61	"Peter"	"german"	29
3	5aa3e8b0bd0c9a4476d06f62	"Tim"	No field	No field

NoSQL Booster Query Tool

https://nosqlbooster.com/downloads



```
db.persons.find({})
```

persons 0.015 s 3 Docs 20

	_id	name	nationality	age	info	info.x
1	ObjectId("5aa3e8b0bd0c9a4476d06f60")	Beat	swiss	35	{ 2 fields }	203
2	ObjectId("5aa3e8b0bd0c9a4476d06f61")	Peter	german	29		
3	ObjectId("5aa3e8b0bd0c9a4476d06f62")	Tim				

DEMO

«spring-data-slides-mongo-examples»

Exercises 23 - 25

https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Part 3:
Validation

Dependencies

```
<dependency>
<groupId>org.hibernate.validator</groupId>
<artifactId>hibernate-validator</artifactId>
<version>6.0.22.Final</version>
</dependency>
```

```
<dependency>
<groupId>org.hibernate.validator</groupId>
<artifactId>hibernate-validator</artifactId>
<version>8.0.0.Final</version>
</dependency>
```

Validation

```
import javax.validation.constraints.AssertTrue;
import javax.validation.constraints.Email;
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.NotNull;
import javax.validation.constraints.Size;

public class User
{
    @NotNull(message = "Name cannot be null")
    private String name;

    @AssertTrue
    private boolean working;

    // ...
}
```

Validation „pure“

```java
public class User
{
    @NotNull(message = "Name cannot be null")
    private String name;

    @AssertTrue
    private boolean working;

    @Size(min = 10, max = 200,
          message = "About Me must be between 10 and 200 characters")
    private String aboutMe;

    @Min(value = 18, message = "Age should not be less than 18")
    @Max(value = 150, message = "Age should not be greater than 150")
    private int age;

    @Email(message = "Email should be valid")
    private String email;

    // standard setters and getters
}
```

Validation „pure“

```
public class ProgramaticValidationExample
{
    public static void main(final String[] args)
    {
        UserWithValidation user = new UserWithValidation();
        user.setWorking(false);
        user.setAboutMe("No info about me!");
        user.setAge(11);

        try (ValidatorFactory factory = Validation.buildDefaultValidatorFactory())
        {
            Validator validator = factory.getValidator();
            Set<ConstraintViolation<UserWithValidation>> violations = validator.validate(user);
            for (ConstraintViolation<UserWithValidation> violation : violations)
            {
                System.err.println(violation.getMessage());
            }
        }
    }
}
```

```
Name cannot be null
muss wahr sein
Age should not be less than 18
```

Validation in JPA

```
@Entity
public class UserWithValidation
{
    @Id @GeneratedValue
    private Long id;

    @NotNull(message = "Name cannot be null")
    private String name;

    @AssertTrue
    private boolean working;

    @Size(min = 10, max = 200,
          message = "About Me must be between 10 and 200 characters")
    private String aboutMe;


    @Min(value = 18, message = "Age should not be less than 18")
    @Max(value = 150, message = "Age should not be greater than 150")
    private int age;

    // ...
    // standard setters and getters
}
```

Validation in JPA

```
private static void executeStatements(final EntityManager entityManager)
{
    UserWithValidation user = new UserWithValidation();
    user.setWorking(false);
    user.setAboutMe("No info about me!");
    user.setAge(11);

    entityManager.persist(user);
    System.out.println(user);
}
```

```
ConstraintViolationImpl{interpolatedMessage='Age should not be less than 18',
propertyPath=age, rootBeanClass=class t_validation.UserWithValidation,
messageTemplate='Age should not be less than 18'}
ConstraintViolationImpl{interpolatedMessage='muss wahr sein', propertyPath=working,
rootBeanClass=class t_validation.UserWithValidation,
messageTemplate='{javax.validation.constraints.AssertTrue.message}'}
ConstraintViolationImpl{interpolatedMessage='Name cannot be null', propertyPath=name,
rootBeanClass=class t_validation.UserWithValidation, messageTemplate='Name cannot be
null'}
```

DEMO

`ProgramaticValidationExample.java`

Validation – Annotations

• `@NotBlank`	can only be applied to text values and validates that the property is not null or blank.
• `@Positive &`	
• `@PositiveOrZero`	are applied to numeric values and validate that they are positive or positive including 0.
• `@Negative &`	
• `@NegativeOrZero`	apply to numeric values and confirm that they are negative or negative including 0.
• `@Past &`	
• `@PastOrPresent`	check whether a date value is in the past or in the past including the present; for all date types including those in Java 8
`@Future &`	
• `@FutureOrPresent`	require that a date value is in the future or in the future including the present.

VALIDATION in Persistence Unit

*** Adjustments in Persistence Unit**

```
<validation-mode>AUTO</validation-mode>

<validation-mode>CALLBACK</validation-mode>
```

- **Pitfalls Versions Hibernate & Hibernate Validator as well as `javax.validation / jakarta.validation`**
    - **Hibernate-Validator 7.x ⇔ / `jakarta.validation`**
    - **Hibernate-Validator 6.x ⇔ / `javax.validation`**

- **Only older variant runs clean in Persistence Unit, otherwise version and initialization problems (*may be possible right now*)**

- **Standalone runs both without problems**

How to build your own validators?

Custom validators in JPA

```
@Target({ElementType.FIELD})
@Retention(RetentionPolicy.RUNTIME)
@Constraint(validatedBy = CheckEnumValidator.class)
public @interface CheckEnum
{
    String message() default "Please enter a valid enum value for this field.";
    Class<?> type();

    // für Constraint
    Class<?>[] groups() default {};
    Class<?extends Payload>[] payload() default {};
}
```

Custom validators in JPA

```
public class CheckEnumValidator implements ConstraintValidator<CheckEnum, String>
{
    Class<?> type;

    @Override
    public void initialize(CheckEnum constraintAnnotation)
    {
        type = constraintAnnotation.type();
        if (!type.isEnum())
            throw new IllegalArgumentException("type is not an enum");
    }

    @Override
    public boolean isValid(String value, ConstraintValidatorContext context)
    {
        ...
    }
}
```

Custom validators in JPA

```
public class CheckEnumValidator implements ConstraintValidator<CheckEnum, String>
{
    ...

    @Override
    public boolean isValid(String value, ConstraintValidatorContext context)
    {
        if (value == null)
            return true;

        Enum<?>[] enumValues = (Enum<?>[])type.getEnumConstants();
        for (Enum<?> enumValue : enumValues)
        {
            if (enumValue.name().equals(value.trim()))
                return true;
        }
        return false;
    }
}
```

Custom validators in JPA

```java
public class ValidatedDomainClass
{
    @NotBlank(message = "Deposit Date is required.")
    @CheckLocalDate(dateFormat = { "yyyy-MM-dd" })
    String depositDate;

    @CheckLocalDate(dateFormat = "dd.MM.yyyy")
    String publicationDate;
    @CheckLocalDate(dateFormat = { "dd.MM.yyyy", "dd.MM.yy" })
    String collectionDate;

    // Enum-Validator for Legacy
    @CheckEnum(type = Seasons.class)
    String season;
    @CheckEnum(type = SpecialColors.class)
    String color;
    @CheckListOfValues(allowedValues = { "Anne", "Will", "Peter", "Lustig" })
    String value;
}
```

DEMO

`CustomValidatorsExample.java`

Validation in Spring

Dependencies

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-validation</artifactId>
</dependency>
```

https://reflectoring.io/bean-validation-with-spring-boot/

Validation

```
import javax.validation.constraints.Max;
import javax.validation.constraints.Min;
import javax.validation.constraints.Pattern;

public class Input {

  @Min(1)
  @Max(10)
  private int numberBetweenOneAndTen;

  @Pattern(regexp = "^[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}\\.[0-9]{1,3}$")
  private String ipAddress;

  // ...
}
```

https://reflectoring.io/bean-validation-with-spring-boot/

Validation

```
@RestController
class ValidateRequestBodyController {

    @PostMapping("/validateBody")
    ResponseEntity<String> validateBody(@Valid @RequestBody Input input) {
        return ResponseEntity.ok("valid");
    }
}
```

https://reflectoring.io/bean-validation-with-spring-boot/

Validation

```
@RestController
@Validated
class ValidateParametersController {

    @GetMapping("/validatePathVariable/{id}")
    ResponseEntity<String> validatePathVariable(@PathVariable("id") @Min(5) int id) {
        return ResponseEntity.ok("valid");
    }

    @GetMapping("/validateRequestParameter")
    ResponseEntity<String> validateRequestParameter(@RequestParam("param") @Min(5) int param)
    {
        return ResponseEntity.ok("valid");
    }

    @ExceptionHandler(ConstraintViolationException.class)
    @ResponseStatus(HttpStatus.BAD_REQUEST)
    ResponseEntity<String> handleConstraintViolationException(ConstraintViolationException e)
{
        return new ResponseEntity<>("not valid due to validation error: " + e.getMessage(),
              HttpStatus.BAD_REQUEST);
    }
}
```

https://reflectoring.io/bean-validation-with-spring-boot/

DEMO

«spring-validation-slides-examples»

DEMO / Hands On

`https://spring.io/guides/gs/validating-form-input/`

Exercise 26

https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Part 4:
Mappings with MapStruct

Mapping

```java
public class Car {

    private String make;
    private int numberOfSeats;
    private CarType type;

    public Car(String make, int numberOfSeats,
               CarType type) {
        this.make = make;
        this.numberOfSeats = numberOfSeats;
        this.type = type;
    }
…
}
```

```java
public class CarDto {

    private String make;
    private int seatCount;
    private String type;
```

```java
public enum CarType {
    PLAIN, PICKUP, SUV, TRUCK
}
```

Are we supposed to transfer every single attribute by hand?

Maven Dependencies & more

To include and activate MapStruct, the POM must be supplemented as follows:

```
<dependency>
    <groupId>org.mapstruct</groupId>
    <artifactId>mapstruct</artifactId>
    <version>1.5.2.Final</version>
</dependency>

<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-compiler-plugin</artifactId>
    <version>3.8.1</version>
    <configuration>
        <release>11</release>
        <annotationProcessorPaths>
            <path>
                <groupId>org.mapstruct</groupId>
                <artifactId>mapstruct-processor</artifactId>
                <version>1.5.2.Final</version>
            </path>
        </annotationProcessorPaths>
    </configuration>
</plugin>
```

Mapping with MapStruct

```
public class SimpleSource {
    private String name;
    private String description;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    …
}
```

```
public class SimpleDestination {
    private String name;
    private String description;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    …
}
```

```
@Mapper
public interface SimpleSourceDestinationMapper {
    SimpleDestination sourceToDestination(SimpleSource source);
    SimpleSource destinationToSource(SimpleDestination destination);
}
```

Mapping with MapStruct

```
public class SimpleSourceDestinationMapperTest {
    private SimpleSourceDestinationMapper mapper =
            Mappers.getMapper(SimpleSourceDestinationMapper.class);

    @Test
    public void sourceToDestinationMapsCorrect() {
        SimpleSource simpleSource = new SimpleSource();
        simpleSource.setName("SourceName");
        simpleSource.setDescription("SourceDescription");

        SimpleDestination destination = mapper.sourceToDestination(simpleSource);

        assertEquals(simpleSource.getName(), destination.getName());
        assertEquals(simpleSource.getDescription(), destination.getDescription());
    }

    …
```

Mapping with MapStruct

```
…

    @Test
    public void destinationToSourceMapsCorrect() {
        SimpleDestination destination = new SimpleDestination();
        destination.setName("DestinationName");
        destination.setDescription("DestinationDescription");

        SimpleSource source = mapper.destinationToSource(destination);

        assertEquals(destination.getName(), source.getName());
        assertEquals(destination.getDescription(), source.getDescription());
    }
}
```

Mapping

```
public class Car {

    private String make;
    private int numberOfSeats;
    private CarType type;

    public Car(String make, int numberOfSeats,
               CarType type) {
        this.make = make;
        this.numberOfSeats = numberOfSeats;
        this.type = type;
    }
…
}
```

```
public class CarDto {

    private String make;
    private int seatCount;
    private String type;
```

```
public enum CarType {
    PLAIN, PICKUP, SUV, TRUCK
}
```

How can we take control?

Mapping with MapStruct

```
@Mapper
public interface CarMapper {

    CarMapper INSTANCE = Mappers.getMapper( CarMapper.class );

    @Mapping(source = "numberOfSeats", target = "seatCount")
    CarDto toDto(Car car);

    @Mapping(source = "seatCount", target = "numberOfSeats")
    Car toCar(CarDto dto);
}
```

Mapping with MapStruct

```
public class CarMapperTest {

    @Test
    public void shouldMapCarToDto() {
        // given
        Car car = new Car("FORD", 5, CarType.PICKUP);

        // when
        CarDto carDto = CarMapper.INSTANCE.toDto(car);

        // then
        assertNotNull(carDto);
        assertEquals("FORD", carDto.getMake());
        assertEquals(5, carDto.getSeatCount());
        assertEquals("PICKUP", carDto.getType());
    }

    …
```

How do you use that in the context of Spring?

Mapping with MapStruct

```java
@Mapper(componentModel = "spring")
public interface ProductMapper {
    ProductDTO toProductDTO(Product product);
    List<ProductDTO> toProductDTOs(List<Product> products);

    Product toProduct(ProductDTO productDTO);
}
```

```java
@Entity
public class Product {
    @Id
    @GeneratedValue
    private Long id;

    private String name;
    private String description;
    private BigDecimal price;

    private Date createdAt;
    private Date updatedAt;
```

```java
public class ProductDTO {
    private String name;
    private String description;
    private BigDecimal price;
```

Mapping with MapStruct

```
@RestController
@RequestMapping("/api/products")
public class ProductAPI {
    private final ProductService productService;
    private final ProductMapper productMapper;

    public ProductAPI(ProductService productService,
                      ProductMapper productMapper) {
        this.productService = productService;
        this.productMapper = productMapper;
    }

    @GetMapping
    @ResponseStatus(HttpStatus.OK)
    public List<ProductDTO> findAll() {
        List<Product> results = productService.findAll();
        return productMapper.toProductDTOs(results);
    }
```

…

Mapping with MapStruct

…

```
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public ProductDTO create(@RequestBody ProductDTO productDTO) {
        Product entity = productMapper.toProduct(productDTO);
        productService.save(entity);

        return productDTO;
    }

    @GetMapping("/{id}")
    public ResponseEntity<ProductDTO> findById(@PathVariable Long id) {
        Optional<Product> optProduct = productService.findById(id);

        if (optProduct.isEmpty())
            return ResponseEntity.notFound().build();
        ProductDTO dto = productMapper.toProductDTO(optProduct.get());
        return ResponseEntity.ok(dto);
    }
```

DEMO

«spring-mapstruct-slides-examples»

Exercise 27

https://github.com/Michaeli71/ADC_BOOTCAMP_SPRING

Questions?

Help

5. Auflage

Michael Inden

Der Weg zum Java-Profi

Konzepte und Techniken für die professionelle Java-Entwicklung

dpunkt.verlag

Michael Inden

Der Java-Profi: Persistenzlösungen und REST-Services

Datenaustauschformate, Datenbankentwicklung und verteilte Anwendungen

dpunkt.verlag

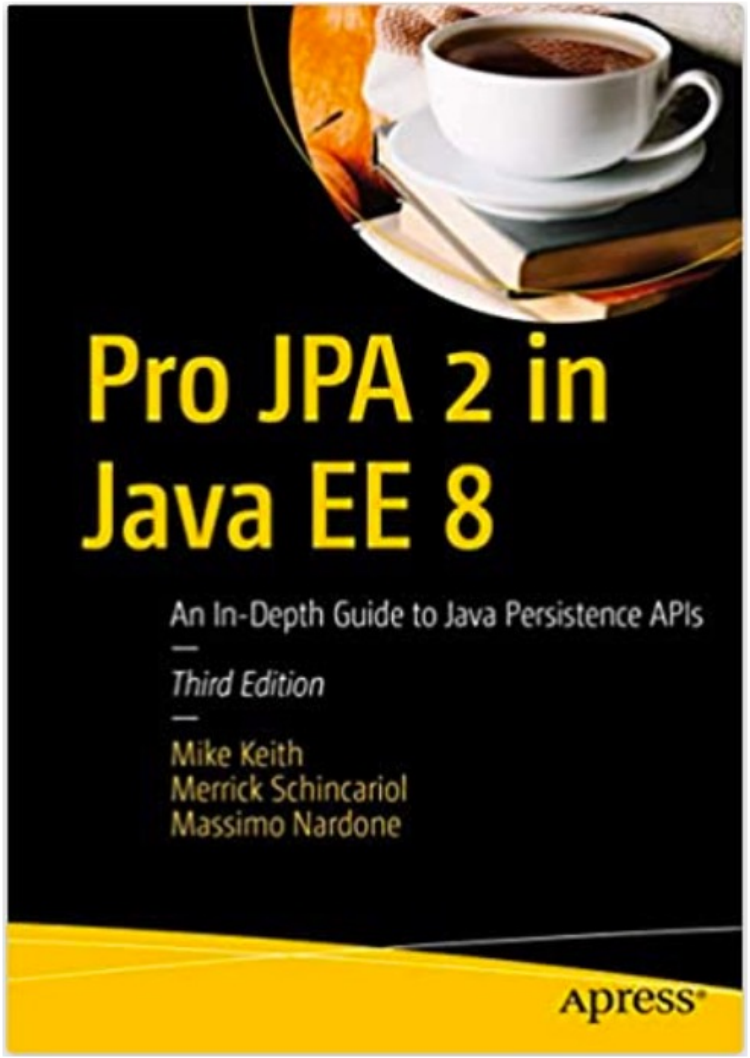
Java Challenge

Michael Inden

Fit für das Job-Interview und die Praxis – mit mehr als 100 Aufgaben und Musterlösungen

dpunkt.verlag

Recommended books



Pro JPA 2 in Java EE 8

An In-Depth Guide to Java Persistence APIs

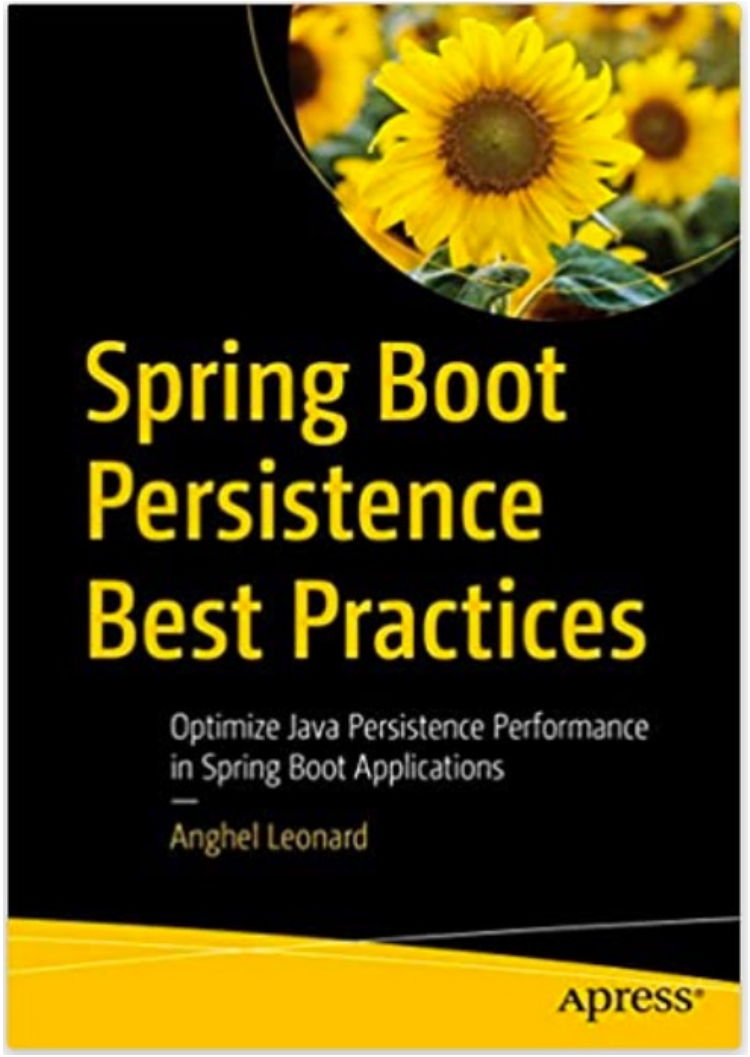
—

Third Edition

—

Mike Keith
Merrick Schincariol
Massimo Nardone

Apress®



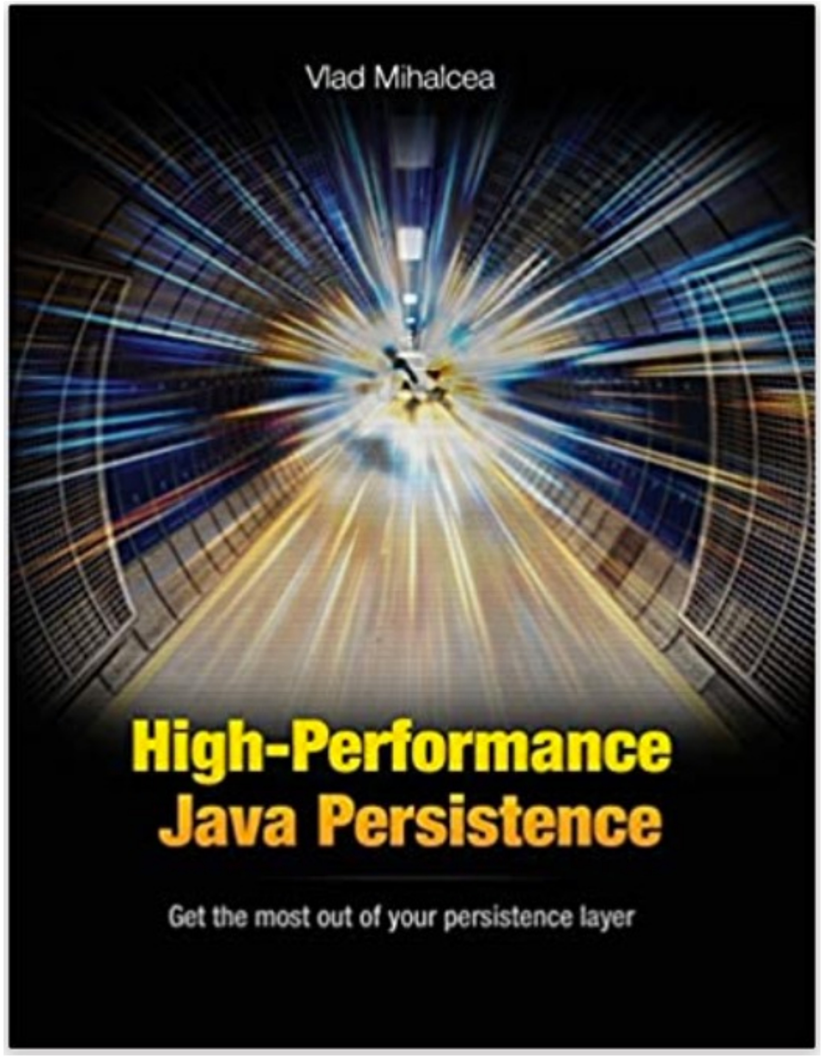
Spring Boot Persistence Best Practices

Optimize Java Persistence Performance in Spring Boot Applications

—

Anghel Leonard

Apress®



Vlad Mihalcea

High-Performance Java Persistence

Get the most out of your persistence layer

Further info / sources

- **ORM**
  - https://thorben-janssen.com/jpa-generate-primary-keys/
  - https://www.objectdb.com/java/jpa/entity/generated
  - https://vladmihalcea.com/orphanremoval-jpa-hibernate/
  - https://www.baeldung.com/jpa-one-to-one
  - https://www.baeldung.com/jpa-cascade-remove-vs-orphanremoval
  - https://www.baeldung.com/hibernate-inheritance
  - https://thorben-janssen.com/complete-guide-inheritance-strategies-jpa-hibernate/
  - https://www.objectdb.com/api/java/jpa/MappedSuperclass
  - https://www.logicbig.com/tutorials/java-ee-tutorial/jpa/mapped-super-class.html
  - https://vladmihalcea.com/the-best-way-to-map-a-onetoone-relationship-with-jpa-and-hibernate/
  - https://www.baeldung.com/jpa-many-to-many
  - https://vladmihalcea.com/the-best-way-to-use-the-manytomany-annotation-with-jpa-and-hibernate/
  - https://stackabuse.com/a-guide-to-jpa-with-hibernate-relationship-mapping/
  - https://thorben-janssen.com/best-practices-for-many-to-many-associations-with-hibernate-and-jpa/

Further info / sources

- **Validation**
  - https://www.baeldung.com/javax-validation
  - https://docs.jboss.org/hibernate/stable/validator/reference/en-US/html_single/

- **MapStruct**
  - https://mapstruct.org/
  - https://www.baeldung.com/mapstruct
  - https://stackabuse.com/guide-to-mapstruct-in-java-advanced-mapping-library/
  - https://www.tutorialspoint.com/mapstruct/index.htm
  - https://auth0.com/blog/how-to-automatically-map-jpa-entities-into-dtos-in-spring-boot-using-mapstruct/
  - https://www.jug.ch/events/slides/190827_Get_smart_with_MapStruct.pdf
  - https://hellokoding.com/mapping-jpa-hibernate-entity-and-dto-with-mapstruct/

Thank You