Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Spring Data JPA

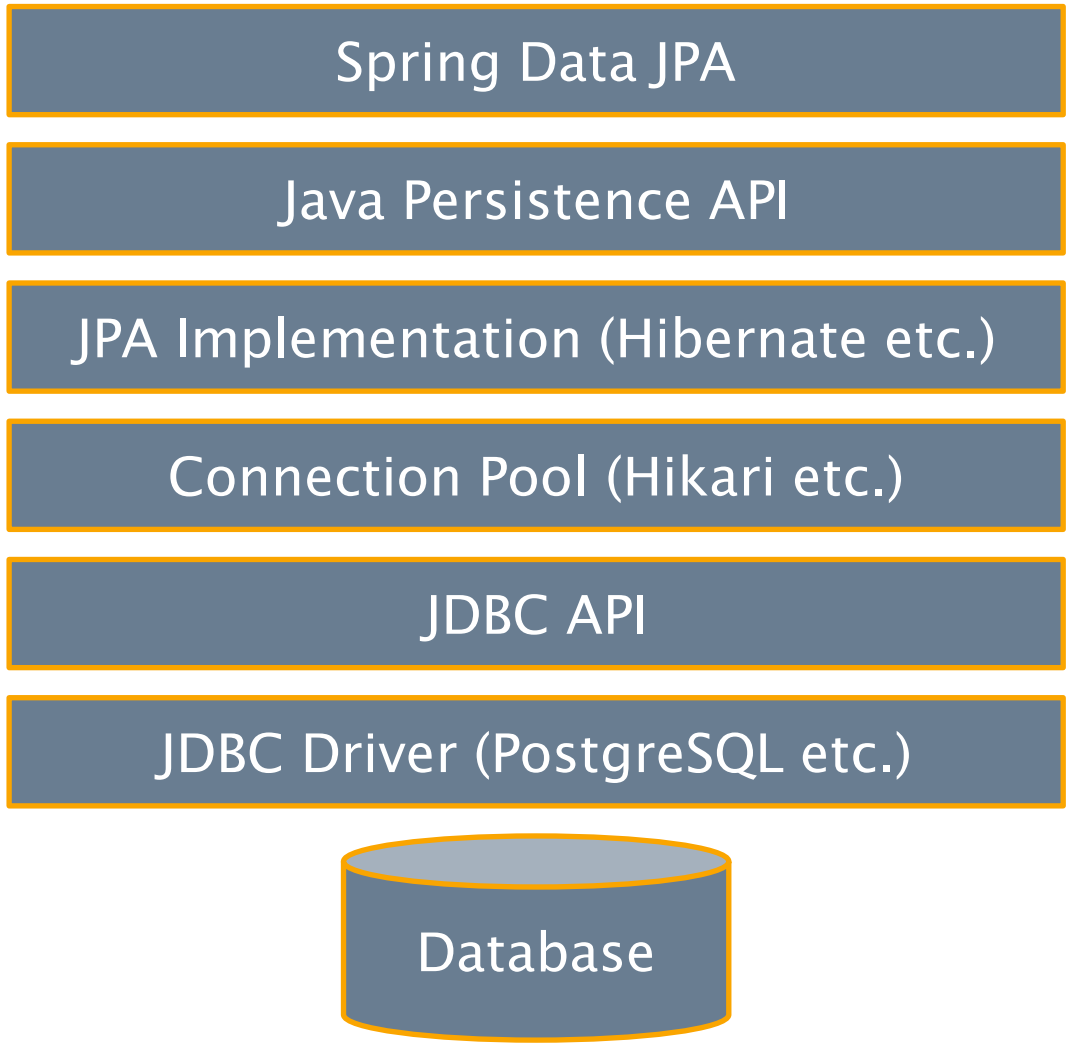
Simon Martinelli

SPRING DATA

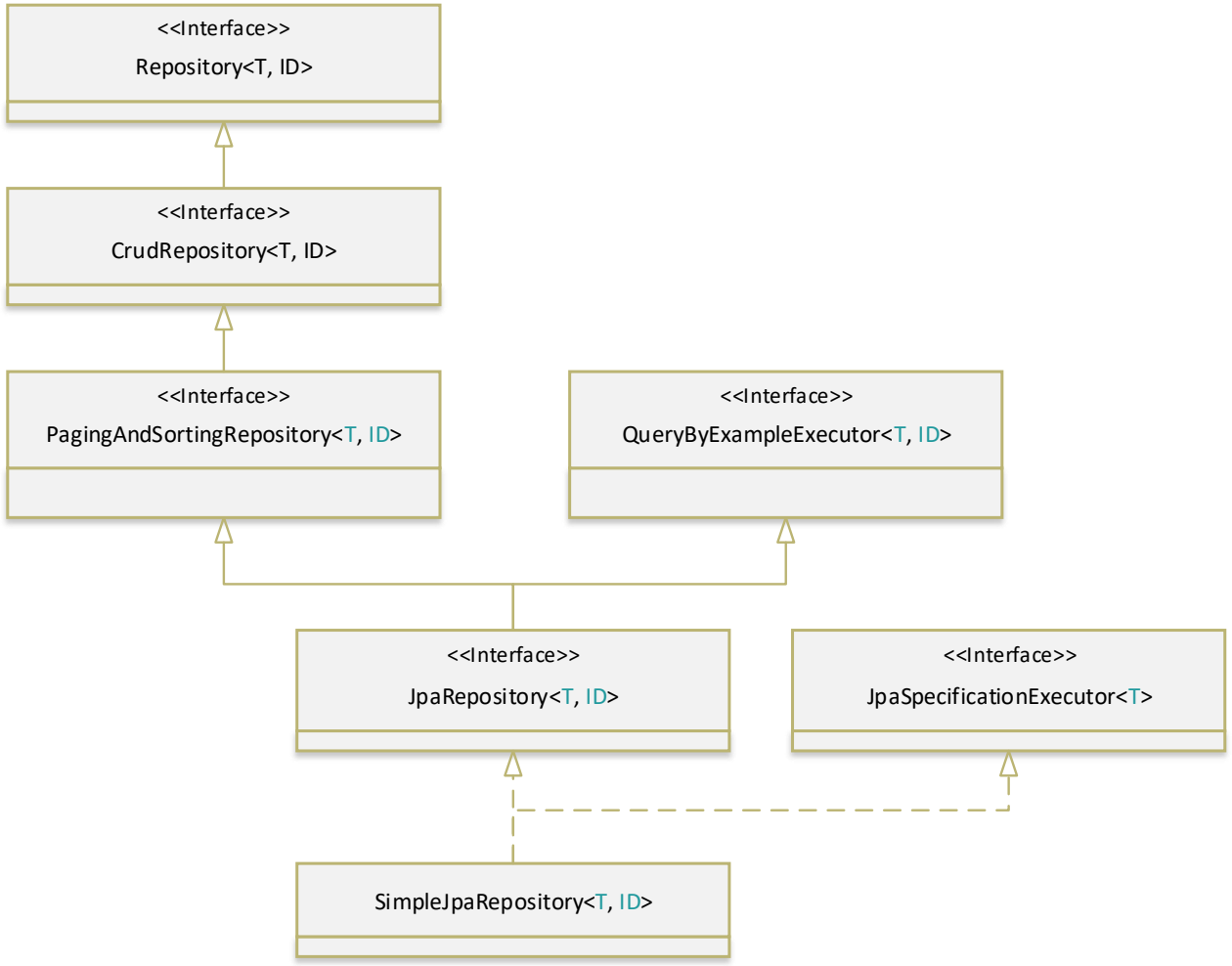
- Spring Data provides a familiar and consistent, Spring-based programming model for data access
- Spring Data facilitates access to relational and non-relational databases, map-reduce frameworks, and cloud-based data services
- Spring Data contains many subprojects that are specific to a given database

Homepage: https://spring.io/projects/spring-data

SPRING DATA TECHNOLOGY STACK



REPOSITORY HIERARCHY



REPOSITORY INTERFACES

- CrudRepository

  `count, delete(All)(ById), existsById, find(All)(ById), save(All)`
- PagingAndSortingRepository

  `findAll`
- JpaRepository

  `delete(All)(ById)InBatch, findAll, flush, save(All)AndFlush`

- The save methods are a combination of the JPA persist and merge operation

- All modifying methods are transactional

REPOSITORY EXAMPLE

- Custom repositories are defined by extending one of the Repository interfaces

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
  ...
}
```

QUERY GENERATION

- Spring Data JPA generates queries based on method names

```
@Repository
public interface EmployeeRepository extends Repository<Employee, Long> {
  Optional<Employee> findByName(String name);
  ...
}
```

- Generated SQL query

```
SELECT e FROM Employee e WHERE e.name = ?1
```

Reference: JPA Query Methods and Repository Query Keywords

QUERY RETURN TYPES

- The find methods can have the following return types:
  - Primitives, wrapper types
  - T, Optional<T>, Collection<T>, List<T>, Stream<T>
  - Future<T>, CompletableFuture<T>
  - Page<T>, Iterable<T>
  - Mono<T>, Flux<T>, Single<T>, Maybe<T>, Flowable<T>

- Methods that return a single instance should use the Optional type to indicate the possible absence of a value

EXPLICIT QUERIES

- Queries can also be defined explicitly using the @Query annotation

```
@Query("select e from Employee e order by e.name")
List<Employee> getEmployees();
```

QUERY PARAMETERS

- Positional parameters

```
@Query("select e from Employee e where e.email = ?1")
Optional<Employee> findEmployee(String email);
```

- Named parameters

```
@Query("select e from Employee e where e.email = :email")
Optional<Employee> findEmployee(String email);
```

DTO BASED PROJECTION

- Constructor expressions can be used to create DTOs

```
@Query("select new hr.dto.EmployeeDTO(e.name, e.department.name) " +
       "from Employee e where e.department = :dept")
List<EmployeeDTO> getEmployees(Department dept);
```

SORTING AND PAGING

- The findAll methods have additional parameters for sorting and paging

```
Iterable<Employee> employees = repository.findAll(Sort.by("name"));

Page<Employee> employees = repository.findAll(PageRequest.of(1, 20));
employees = repository.findAll(PageRequest.of(1, 20, Sort.by("name")));
```

MODIFYING QUERIES

- Insert, update and delete queries must be annotated as modifying queries

```
@Modifying
@Query("update Employee e set e.salary = ?1 where e.salary = ?2")
int updateSalary(long newSalary, long oldSalary);
```

DATASOURCE CONFIGURATION

- Spring application properties can be used to configure the data source

```
spring.datasource.url=jdbc:postgresql://localhost:5432/hr
spring.datasource.username=postgres
spring.datasource.password=postgres
```

DATABASE INITIALIZATION

- A Hibernate feature can be used to generate the database schema automatically (default value is create-drop for embedded databases, none otherwise)

```
spring.jpa.hibernate.ddl-auto=validate
```

- Alternatively, the scripts `schema.sql` and `data.sql` (located in the classpath) can be used to initialize the database with a schema and data

- Database initialization can be activated with an application property (activated by default only for embedded databases)

```
spring.sql.init.mode=always
```

TESTING

- The @DataJpaTest annotation is used to test the persistence layer of Spring Boot applications (test slice)
  - configures an in-memory embedded database
  - scans for @Entity classes
  - loads JPA repositories (but not other components)

- Example

```
@DataJpaTest
public class EmployeeRepositoryTest { ... }
```