Mapowanie obiektowo – relacyjne

Marcin Nowak

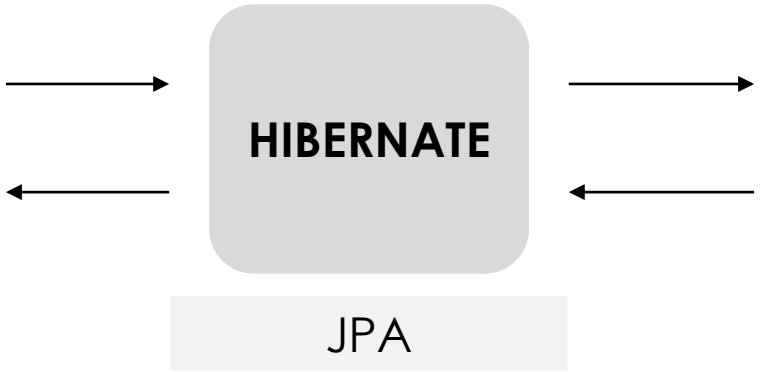
> Więcej niż połowa problemów wydajnościowych aplikacji ma swoje podłoże w bazie danych.
>
> - https://www.appdynamics.com/supported-technologies/database

Agenda

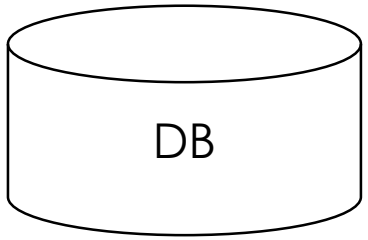
1. JPA
2. Mapowania ORM
3. Spring Data

JPA i Hibernate

Article	
id:	Long
author:	String
createdOn:	LocalDateTime
title:	String
content:	String
tag:	Tag

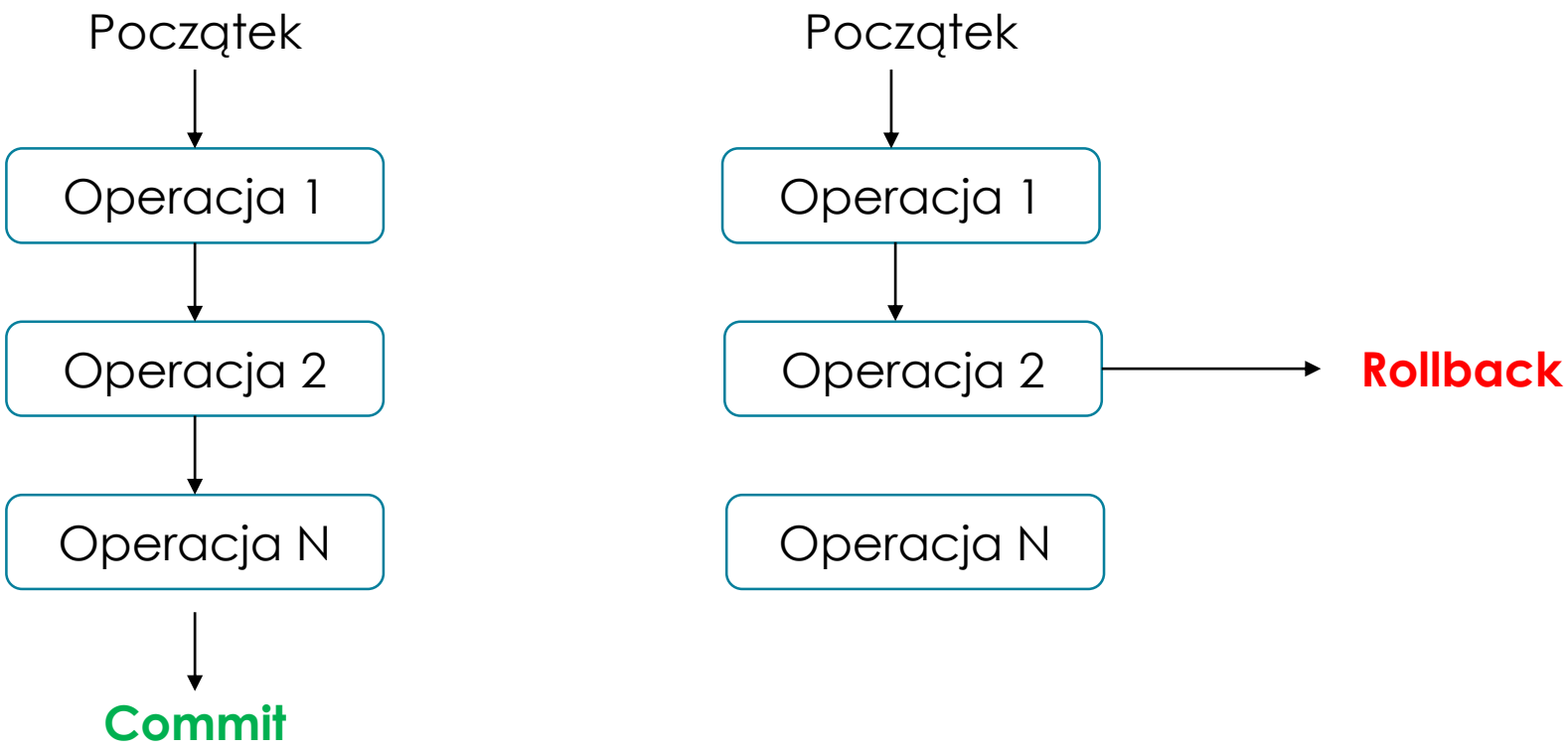


ARTICLE	
ID	number(38,0)
AUTHOR	varchar2(20 chars)
CREATED_ON	timestamp(6)
TITLE	varchar2(25 chars)
TAG_ID	number(38,0)
CONTENT	varchar2(1000 chars)



Transakcje

Atomowość | **Spójność** | **Izolacja** | **Trwałość**

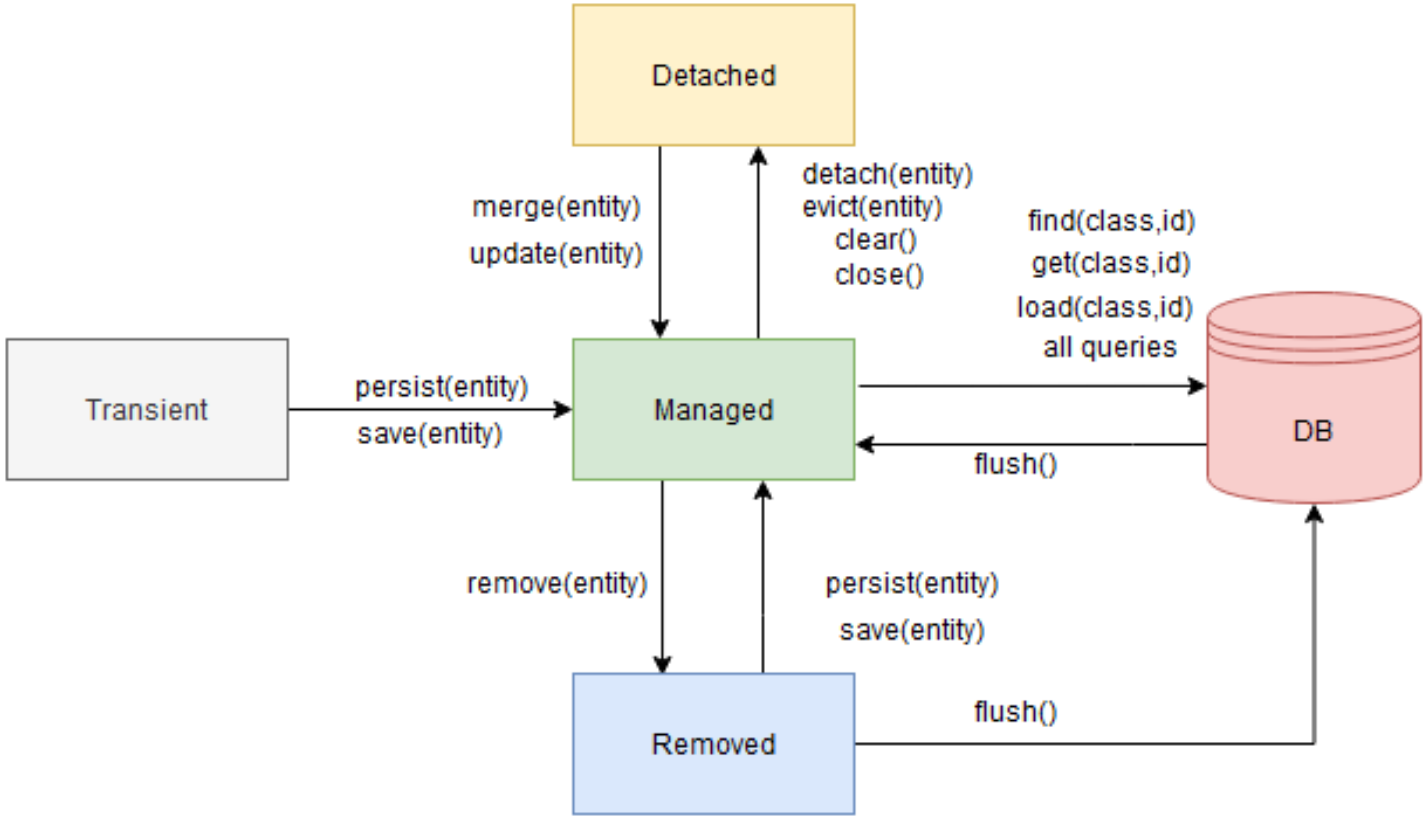


Entity Manager

```
<bean id="applicationEntityManagerFactory" class="pl.wroc.pwr.example.ConfigurableContainerEntityManagerFactoryBean">
    <property name="dataSource" ref="applicationDataSource" />
    <property name="persistenceUnitName" value="applicationPersistenceUnit" />
    <property name="defaultFlushMode" value="COMMIT" />
    <property name="persistenceXmlLocation" value="classpath:META-INF/persistence.xml" />
     ...
</bean>
```

```
EntityManagerFactory entityManagerFactory =
        Persistence.createEntityManagerFactory("applicationPersistenceUnit");
EntityManager em = entityManagerFactory.createEntityManager();

// wczytanie encji o ID = 123
Customer customer = em.find(Customer.class, 123L);

// usunięcie encji
em.remove(customer);

// utrwalenie nowej encji
Customer newCustomer = new Customer(123L, "Bosch Polska");
em.persist(newCustomer);
```

JPA - cykl życia encji



JPA Persist


```
Product product = new Product();
product.setName("Sony Bravia OLED 55 TV"); //transient

em.getTransaction().begin();
em.persist(product);
em.getTransaction().commit();
```

Generowanie kluczy podstawowych

GenerationType.IDENTITY

GenerationType. SEQUENCE

GenerationType.TABLE

GenerationType.AUTO

GenerationType.IDENTITY

```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long id;
```

```
INSERT INTO PRODUCT (id, name)
VALUES (DEFAULT, 'Sony Bravia OLED 55 TV')
```

GenerationType.SEQUENCE

```
@Id
@GeneratedValue(
    strategy = GenerationType.SEQUENCE,
    generator = "custom_sequence"
)
@SequenceGenerator(
    name = "custom_sequence",
    allocationSize = 2
)
private Long id;
```

```
CREATE SEQUENCE custome_sequence START 1 INCREMENT 2
```

Update – encja managed

```
EntityManager em = entityManagerFactory.createEntityManager();
em.getTransaction().begin();

Product product = em.find(Product.class, id);
product.setName("Nowa nazwa");

em.getTransaction().commit();
em.close();
```

Update – encja detached

```
EntityManager em = entityManagerFactory.createEntityManager();

Product product = new Product();
product.setName("Iphone 12");

em.persist(product);

//em is closed

product.setName("Samsung Galaxy S21");

Product product = em.merge(product);
```

Modelowanie encji

```
@Entity
@Table(name = "LIBRARY")
public class LibraryEntity {
    @Id
    private Long id;
    @Column(name = "NAME", length = 30, nullable = false)
    private String name;
    @Column(name = "DOMAIN", length = 5, nullable = true)
    private String domain;

    public LibraryEntity() {
    }

    // getters and setters
}
```

Strategie dziedziczenie

- Mapped superclass
- Single Table
- Join Table
- Table per class

@MappedSuperclass

```java
@Entity
@Table(name = "order_notification")
public class OrderNotification
        extends Notification {
    @Column(
            name = "price",
            nullable = false
    )
    private BigDecimal totalPrice;

    //Getters and setters
}
```

```java
@MappedSuperclass
public abstract class Notification {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "sender_name")
    private String senderName;

    @Temporal( TemporalType.TIMESTAMP )
    @CreationTimestamp
    @Column(name = "created_on")
    private Date createdOn;

    //Getters and setters
}
```

```java
@Entity
@Table(name = "product_notification")
public class ProductNotification
        extends Notification {

    @Column(
            name = "available",
            nullable = false
    )
    private Boolean inStock;

    //Getters and setters
}
```

order_notification
id number(38,0)
sender_name varchar2(20 chars)
created_on timestamp(6)
price number(15,3)

product_notification
id number(38,0)
sender_name varchar2(20 chars)
created_on timestamp(6)
available number(1,0)

Table per class

```java
@Entity
public class OrderNotification
        extends Notification {
    @Column(
            name = "price",
            nullable = false
    )
    private BigDecimal totalPrice;

    //Getters and setters
}
```

```java
@Entity
@Inheritance(
     strategy = InheritanceType.TABLE_PER_CLASS)
public class Notification {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "sender_name")
    private String senderName;

    @Temporal( TemporalType.TIMESTAMP )
    @CreationTimestamp
    @Column(name = "created_on")
    private Date createdOn;

    //Getters and setters
}
```

```java
@Entity
public class ProductNotification
        extends Notification {

    @Column(
            name = "available",
            nullable = false
    )
    private Boolean inStock;

    //Getters and setters
}
```

order_notification
id number(38,0)
sender_name varchar2(20 chars)
created_on timestamp(6)
price number(15,3)

notification
id number(38,0)
sender_name varchar2(20 chars)
created_on timestamp(6)

product_notification
id number(38,0)
sender_name varchar2(20 chars)
created_on timestamp(6)
available number(1,0)

Single Table

```java
@Entity
@DiscriminatorValue("ON")
public class OrderNotification
        extends Notification {
    @Column(
            name = "price",
            nullable = false
    )
    private BigDecimal totalPrice;

    //Getters and setters
}
```

```java
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(length = 2)
public abstract class Notification {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "sender_name")
    private String senderName;

    @Temporal( TemporalType.TIMESTAMP )
    @CreationTimestamp
    @Column(name = "created_on")
    private Date createdOn;

    //Getters and setters
}
```

```java
@Entity
@DiscriminatorValue("PN")
public class ProductNotification
        extends Notification {

    @Column(
            name = "available",
            nullable = false
    )
    private Boolean inStock;

    //Getters and setters
}
```

notification
id number(38,0)
dtype varchar2(2 char)
sender_name varchar2(20 chars)
created_on timestamp(6)
available number(1,0)
price number(15,3)

Joined

```java
@Entity
@DiscriminatorValue("ON")
public class OrderNotification
        extends Notification {
    @Column(
            name = "price",
            nullable = false
    )
    private BigDecimal totalPrice;

    //Getters and setters
}
```

```java
@Entity
@Inheritance(
     strategy = InheritanceType.JOINED)
@DiscriminatorColumn(length = 2)
public class Notification {

    @Id
    @GeneratedValue
    private Long id;

    @Column(name = "sender_name")
    private String senderName;

    @Temporal( TemporalType.TIMESTAMP )
    @CreationTimestamp
    @Column(name = "created_on")
    private Date createdOn;

    //Getters and setters
}
```

```java
@Entity
@DiscriminatorValue("PN")
public class ProductNotification
        extends Notification {

    @Column(
            name = "available",
            nullable = false
    )
    private Boolean inStock;

    //Getters and setters
}
```

order_notification
id number(38,0)
price number(15,3)

notification
id number(38,0)
sender_name varchar2(20 chars)
created_on timestamp(6)
dtype varchar2(2 char)

product_notification
id number(38,0)
available number(1,0)

Relacje

@ManyToOne

@OneToMany

@OneToOne

@ManyToMany

@OneToMany

```java
@Entity
public class Article{
    @Id
    @GeneratedValue
    private Long id;
    private String author;
    private LocalDateTime createdOn;
    private String title;
    private String content;

    @OneToMany(cascade = CascadeType.ALL, orphanRemoval = true)
    @JoinColumn(name = "article_id")
    private List<Comment> comments;
}
```

```java
@Entity
public class Comment{
    @Id
    @GeneratedValue
    private Long id;
    private String text;
}
```

Article	
id	Long
author	String
createdOn	LocalDateTime
title	String
content	String
comments	List<Comment>

Comment	
id	Long
text	String

article	1	*	comment
id number(38,0)			**id** number(38,0)
author varchar2(20 chars)			**text** varchar2(50 chars)
created_on timestamp(6)			**article_id** number(38,0)
title varchar2(25 chars)			
content varchar2(1000 chars)			

@OneToOne

```java
@Entity
public class Article{
    @Id
    @GeneratedValue
    private Long id;
    private String author;
    private LocalDateTime createdOn;
    private String title;
    private String content;

    @OneToOne
    @JoinColumn(name = "tag_id")
    private Tag tag;
}
```

```java
@Entity
public class Tag{
    @Id
    @GeneratedValue
    private Long id;
    private String name;
}
```

Article	
id	Long
author	String
createdOn	LocalDateTime
title	String
content	String
tag	Tag

1

1

Tag	
id	Long
name	String

article
id number(38,0)
author varchar2(20 chars)
created_on timestamp(6)
title varchar2(25 chars)
tag_id number(38,0)
content varchar2(1000 chars)

1 — 1

tag
id number(38,0)
name varchar2(20 chars)

@ManyToOne

```java
@Entity
public class Article{
    @Id
    @GeneratedValue
    private Long id;
    private String author;
    private LocalDateTime createdOn;
    private String title;
    private String content;

    @ManyToOne
    @JoinColumn(name = "tag_id")
    private Tag tag;
}
```

```java
@Entity
public class Tag{
    @Id
    @GeneratedValue
    private Long id;
    private String name;
}
```

Article	
id	Long
author	String
createdOn	LocalDateTime
title	String
content	String
tag	Tag

*

1

Tag	
id	Long
name	String

article
id number(38,0)
author varchar2(20 chars)
created_on timestamp(6)
title varchar2(25 chars)
tag_id number(38,0)
content varchar2(1000 chars)

*

1

tag
id number(38,0)
name varchar2(20 chars)

@ManyToMany

```
@Entity
public class Article{
    @Id
    @GeneratedValue
    private Long id;
    private String author;
    private LocalDateTime createdOn;
    private String title;
    private String content;

    @ManyToMany
    @JoinTable(name = "article_tag_X",
        joinColumns = @JoinColumn(name ="article_id"),
        inverseJoinColumns = @JoinColumn(name = "tag_id"))
    private Set<Tag> tags;
}
```

```
@Entity
public class Tag{
    @Id
    @GeneratedValue
    private Long id;
    private String name;
}
```

Article	
id	Long
author	String
createdOn	LocalDateTime
title	String
content	String
tags	Set<Tag>

* → *

Tag	
id	Long
name	String

article
id number(38,0)
author varchar2(20 chars)
created_on timestamp(6)
title varchar2(25 chars)
content varchar2(1000 chars)

* — *

article_tag_x
article_id number(38,0)
tag_id number(38,0)

* — *

tag
id number(38,0)
name varchar2(20 chars)

Pobieranie danych zależnych - FetchType

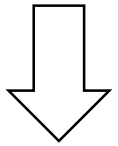
FetchType.EAGER

FetchType. LAZY

FetchType.EAGER

```
@Entity
public class Article{
    @Id
    @GeneratedValue
    private Long id;
    private String author;
    private LocalDateTime createdOn;
    private String title;
    private String content;

    @ManyToOne
    private Tag tag;
}
```

```
Article article = entityManager.find(Article.class, 1L)
```

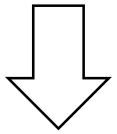


```
SELECT *
FROM article a
LEFT OUTER JOIN tag t ON a.tag_id = t.id
WHERE a.id = 1
```

FetchType.LAZY

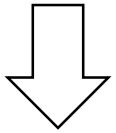
```java
@Entity
public class Article{
    @Id
    @GeneratedValue
    private Long id;
    private String author;
    private LocalDateTime createdOn;
    private String title;
    private String content;

    @OneToMany
    private List<Comment> comments;
}
```

```java
Article article = entityManager.find(Article.class, 1L)
```



```
SELECT a.*
FROM article a
WHERE a.id = 1
```

```java
int commentsAmount = article.getComments().size();
```



```
SELECT *
FROM article a
LEFT OUTER JOIN tag t ON a.tag_id = t.id
WHERE a.id = 1
```

Problem N+1

```java
@Entity
public class Tweet {
    @Id
    @GeneratedValue
    private Long id;
    @NotNull
    private String content;
    private Integer retweetCount;
    @NotNull
    private User author;
    //...
}
```

```java
@Entity
public class User {
    @Id
    @GeneratedValue
    private Long id;
    @NotNull
    private String userName;
    private Integer followersCount;
    //...
}
```

```java
List<Tweet> tweets = entityManager.createQuery(
    "select t from Tweet t where t.retweetCount > 10", Tweet.class
    )
    .getResultList();

LOGGER.info("Ile takich tweetow: "+tweets.size());

for(Tweet tweet: tweets){
    LOGGER.info("Autorem popularnego tweeta jest: "+tweet.getAuthor().getUserName());
}
```

Problem N+1 - rozwiązanie

```
List<Tweet> tweets = entityManager.createQuery(
    "select t from Tweet t "+
    "join fetch t.author u "+
    "where t.retweetCount > 10", Tweet.class
    )
    .getResultList();
```

```
SELECT ...
FROM tweet t
INNER JOIN user u ON t.author_id = u.id
WHERE retweetCount > 10
```

Optimistic lock

```
@Entity
public class Article{
    @Id
    @GeneratedValue
    private Long id;
    private String author;
    private LocalDateTime createdOn;
    private String title;
    private String content;

    @Version
    private long version;
}
```

OptimisticLockException

Jan: `Article article = entityManager.find(Article.class, 1L)`

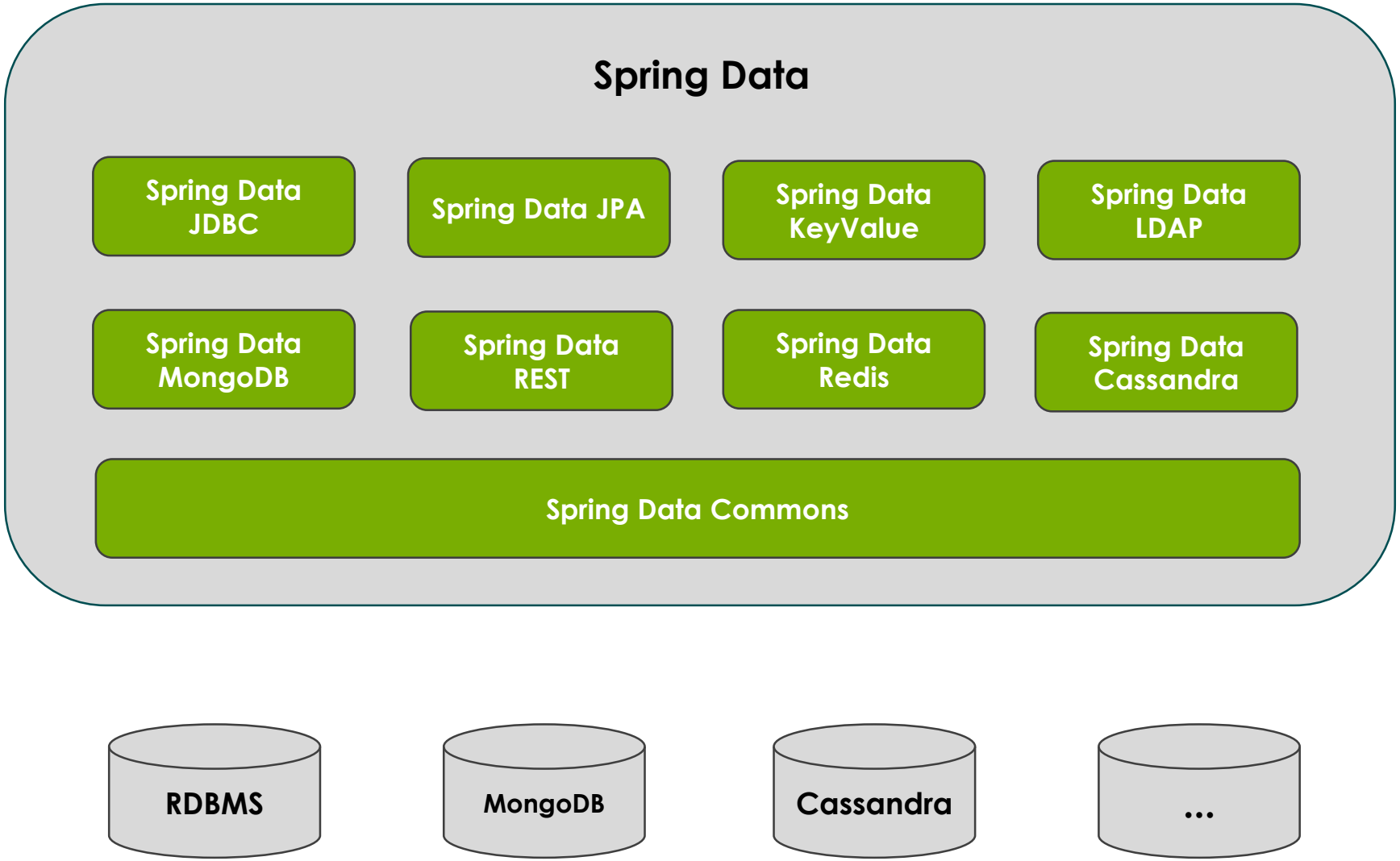
Jan: `article.setTitle("Inny tytul");`

Kasia: `Article article = entityManager.find(Article.class, 1L)`

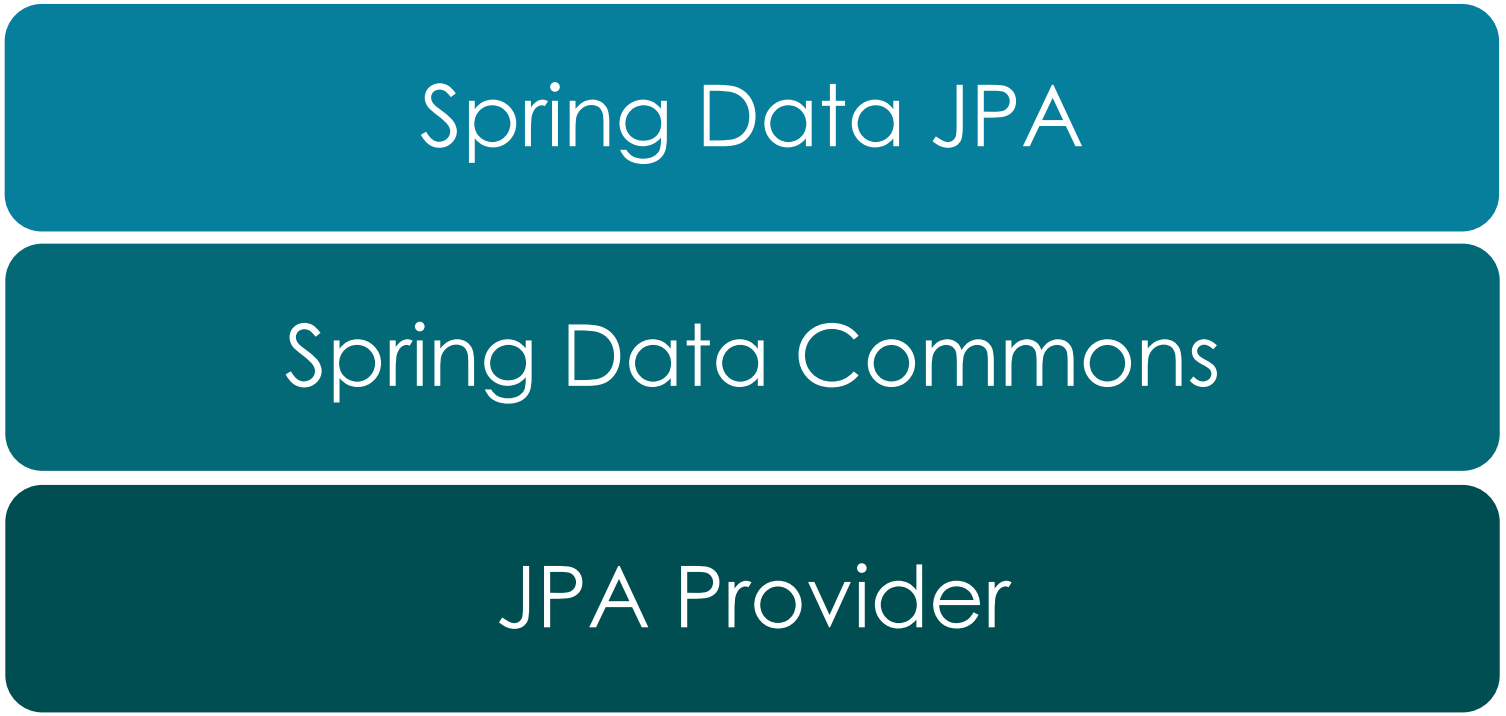
Kasia: `article.setTitle("Nowy tytul");`

Czas

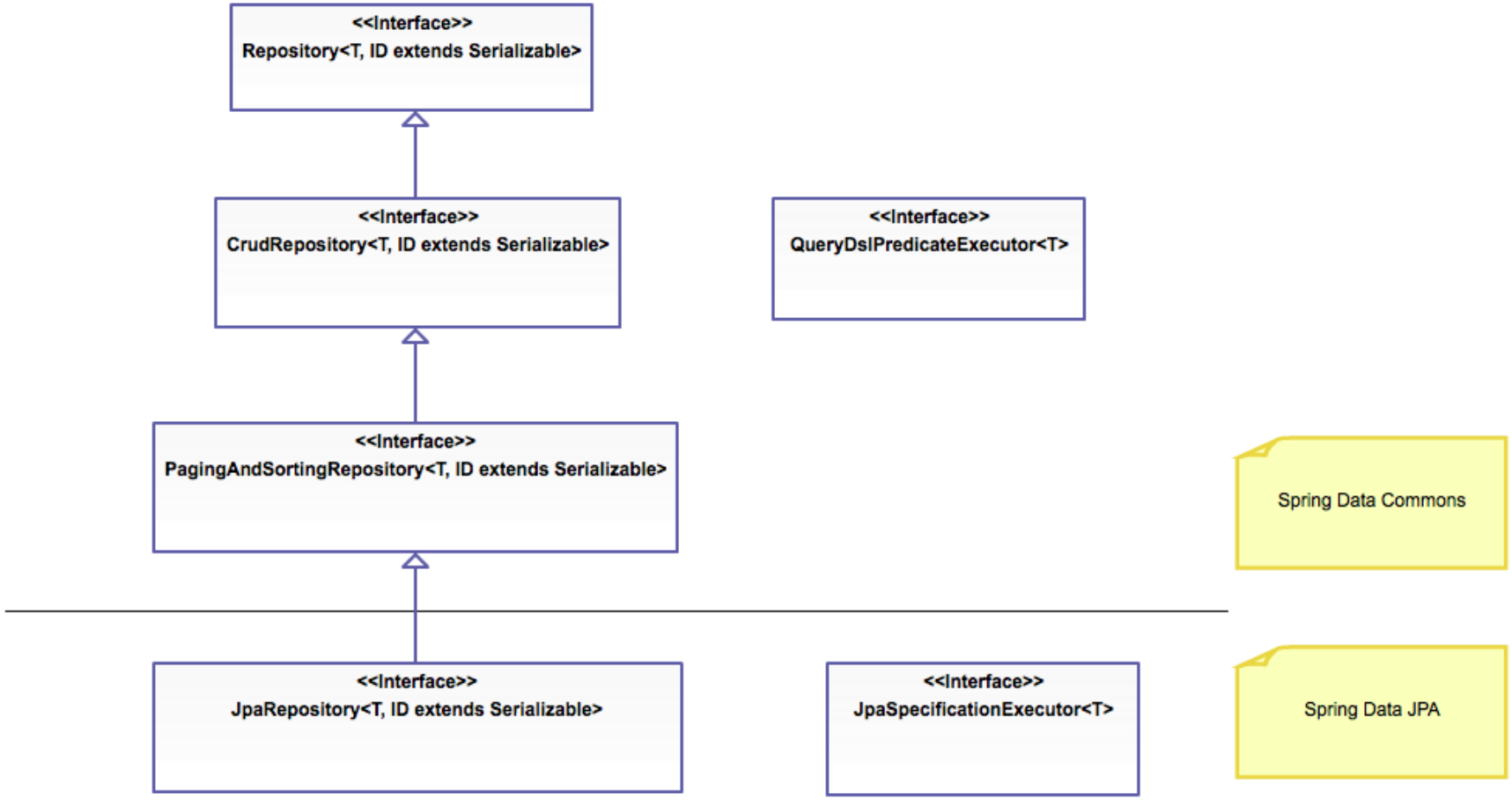
Spring Data


Spring Data
Spring Data JDBC
Spring Data JPA
Spring Data KeyValue
Spring Data LDAP
Spring Data MongoDB
Spring Data REST
Spring Data Redis
Spring Data Cassandra
Spring Data Commons
RDBMS
MongoDB
Cassandra
...

Spring Data JPA



Spring Data - repozytoria



Tradycyjne JPA – przykład

```
@Repository
@Transactional
class JpaCustomerRepository implements CustomerRepository {
    @PersistenceContext
    private EntityManager em;

    @Override
    @Transactional
    public Customer save(Customer customer) {
        if (customer.getId() == null) {
            em.persist(customer);
            return customer;
        }
        else {
            return em.merge(customer);
        }
    }

    @Override
    public Customer findByName(String name) {
        TypedQuery<Customer> query = em.createQuery(
                "select c from Customer c where c.name = :name", Customer.class);
        query.setParameter("name", name);
        return query.getSingleResult();
    }
}
```

Spring Data – przykład

```
package com.companyname.component;

public interface CustomerRepository extends CrudRepository<Customer, Long> {

    Customer findByName(String name);

    Customer save(Customer customer);
}
```

```
xmlns:jpa="http://www.springframework.org/schema/data/jpa"
<jpa:repositories base-package="com.companyname.component" />
```

Spring Data – jak to działa?

```
public interface CustomerRepository extends CrudRepository<Customer, Long> {
    Customer findByName(String Name);
    List<Customer> findByEmailAndLastname(String email, String lastname);
    List<Customer> findByAddressZipCode(String zipCode);
}
```

Słowo kluczowe	Przykład	Zapytanie
Distinct	findDistinctByLastnameAndFirstname	select distinct … where x.lastname = ? and x.firstname = ?
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?
Like	findByFirstnameLike	… where x.firstname like ?
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ? order by x.lastname desc
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?

Spring Data – paginacja i sortowanie

```
Page<Customer> findByLastname(String lastname, Pageable pageable);
List<Customer> findByLastname(String lastname, Sort sort);
List<Customer> findByLastname(String lastname, Pageable pageable);
```

```
Pageable pageable = new PageRequest(2, 10,
                    Direction.ASC,
                    "lastname", "firstname");

Page<Customer> result = findByLastname(„Kowalski", pageable);

Sort sort = new Sort(Direction.DESC, "lastname", "firstname");
List<Customer> result = findByLastname(„Kowalski", sort);
```

Criteria Query API

```
@Override
public List<Customer> findAllCustomers() {
    CriteriaBuilder cb = entityManager.getCriteriaBuilder();
    CriteriaQuery<Customer> query = cb.createQuery(Customer.class);
    Root<Customer> root = query.from(Customer.class);
    query.select(root);

    TypedQuery<Customer> typedQuery = entityManager.createQuery(query);

    return typedQuery.getResultList();
}
```

Query DSL

```
EntityManager em = entityManagerFactory.createEntityManager();
JPAQueryFactory queryFactory = new JPAQueryFactory(em);

QCustomer customer = QCustomer.customer;

Customer c = queryFactory
        .selectFrom(customer)
        .where(customer.name.eq("Pepsi Co. PL"))
        .fetchOne();
```

Dziękuję za uwagę