Spring Data vs JPA

10 real-life stories

December, 9th 2020

- Sergiy Morenets, 2020

DEVELOPER 16 YEARS

TRAINER 7 YEARS

WRITER 4 BOOKS

- Sergiy Morenets, 2020

FOUNDER

ITSimulator

SPEAKER

JAVA DAY
MINSK 2013

Dev(Talks):

Java
frameworks
day

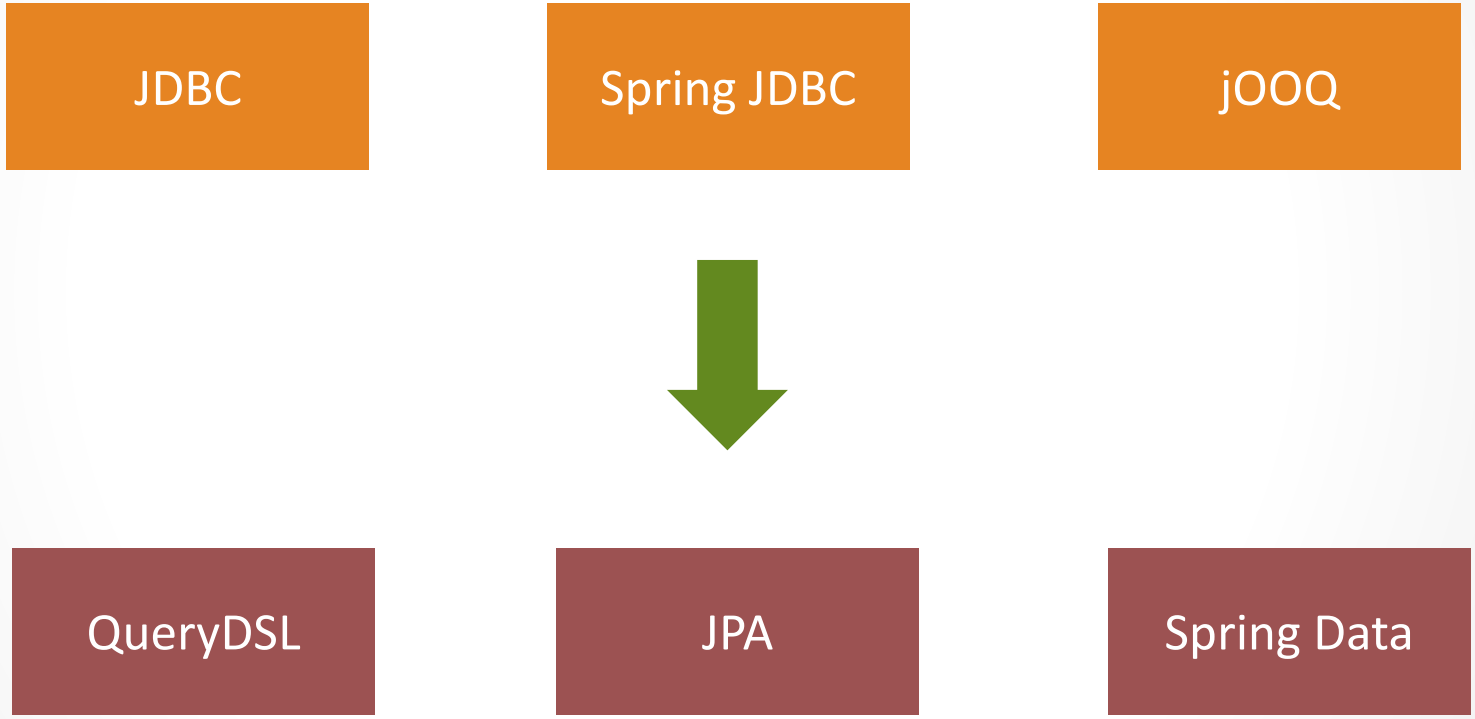
Standard webinar

- Sergiy Morenets, 2020

Agenda

- ✓ ORM and relational mapping. Technologies
- ✓ Spring Data and JPA overview
- ✓ API usage
- ✓ Queries and CRUD operations
- ✓ Performance & speed

Relational data mapping



ORM. Mapped entities


```
@Getter @Setter
@Entity @Table
public class Product {
    @Id
    private int id;

    private String name;

    private Set<Order> orders;
```

JPA

- Part of **J2EE** (Jakarta EE)
- Abstracts mapping in ORM systems
- JPA **1.0** was introduced in May 2006
- JPA **2.0** released in December 2009
- JPA **2.1** released in April 2013
- JPA **2.2** released in June 2017
- Introduces JPQL
- Renamed to **Jakarta Persistence** in 2019
- JPA 3.0 released in 2020

javax.persistence ⟶ jakarta.persistence

- Sergiy Morenets, 2020

Spring Data

- Started in 2008
- Reduces programming effort and avoids boilerplate code
- Db-agnostic repository abstraction
- Dynamic query construction
- Spring Data JPA is abstraction layer over JPA/Spring ORM/Spring Tx
- Includes integration with JDBC/Mongo/Redis/REST/ Cassandra/Couchbase/ElasticSearch/Hadoop/Neo4j
- Simpler REST development with **Spring Data Rest**

Spring Data Modules

Core modules	Core	JPA	MongoDB
	Neo4j	Solr	Gemfire
	REST	Redis	KeyValue
Community modules	Couchbase	Elasticsearch	Cassandra

Case #1. Configuration

JPA. XML configuration

```
<persistence version="2.2"
    xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
    http://xmlns.jcp.org/xml/ns/persistence/persistence_2_2.xsd">
    <persistence-unit name="school" transaction-type="RESOURCE_LOCAL">
        <provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
        <class>org.sample.it.migration.Order</class>
        <class>org.sample.it.migration.Product</class>
        <exclude-unlisted-classes>true</exclude-unlisted-classes>
        <properties>
            <property name="hibernate.dialect" value="org.hibernate.dialect.
            <property name="hibernate.hbm2ddl.auto" value="update" />
            <property name="hibernate.connection.driver_class" value="org.hs
            <property name="hibernate.connection.url" value="jdbc:hsqldb:mem
```

Hibernate is able to detect entity classes in classpath

persistence.xml

JPA. Spring configuration

- ✓ Enable transaction management
- ✓ Declare transaction manager bean
- ✓ Declare entity manager factory bean
- ✓ Declare properties factory bean
- ✓ Specify datasource settings

JPA. Spring Boot configuration

```
spring.jpa.database-platform=H2
```

```yaml
spring:
  jpa:
    database: h2
    properties:
      hibernate:
        ddl-auto: update
        cache:
          region:
            factory_class: org.hibernate.cache.jcache.JCacheRegionFactory
          use_query_cache: true
        jdbc:
          batch_size: 20
```

Spring Data JPA. Configuration

```
@SpringBootApplication
public class MainApplication {

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

Enables Spring Data JPA auto-configuration

Custom configuration (optional)

```
@SpringBootApplication
@EntityScan("it.discovery.persistence.model")
@EnableJpaRepositories(basePackages = "persistence.data",
                       bootstrapMode = BootstrapMode.DEFERRED)
public class MainApplication {
```

Also DEFAULT, LAZY

Defer Spring Data initialization till application context bootstrap complete

Case #2. CRUD operations

POST /products

JPA. Operation template


```
public static void main(String[] args) {
    EntityManagerFactory emf = Persistence
            .createEntityManagerFactory("shop");
    EntityManager em = null;
    try {
        em = emf.createEntityManager();

        em.getTransaction().begin();

        em.getTransaction().commit();
    } catch (Exception ex) {
        ex.printStackTrace();
    } finally {
        em.close();
        emf.close();
    }
```

Any operation should be wrapper into a transaction

Spring + JPA. API usage

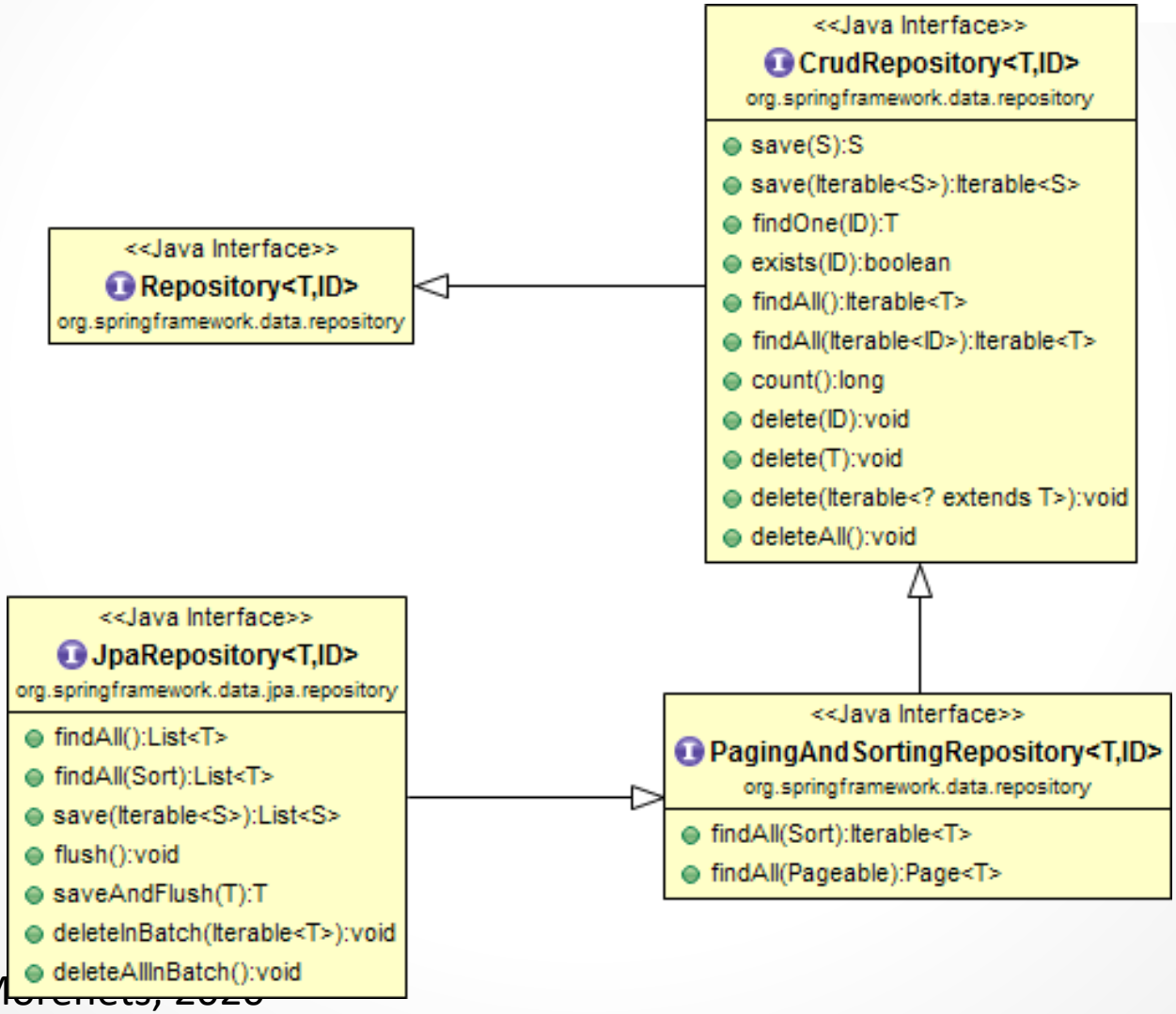

```
public interface ProductRepository {

    void save(Product p);
```

```
@Repository
public class SpringJpaProductRepository implements ProductRepository {

    @PersistenceContext
    private EntityManager em;

    @Override
    public void save(Product p) {
        em.persist(p);
    }
```

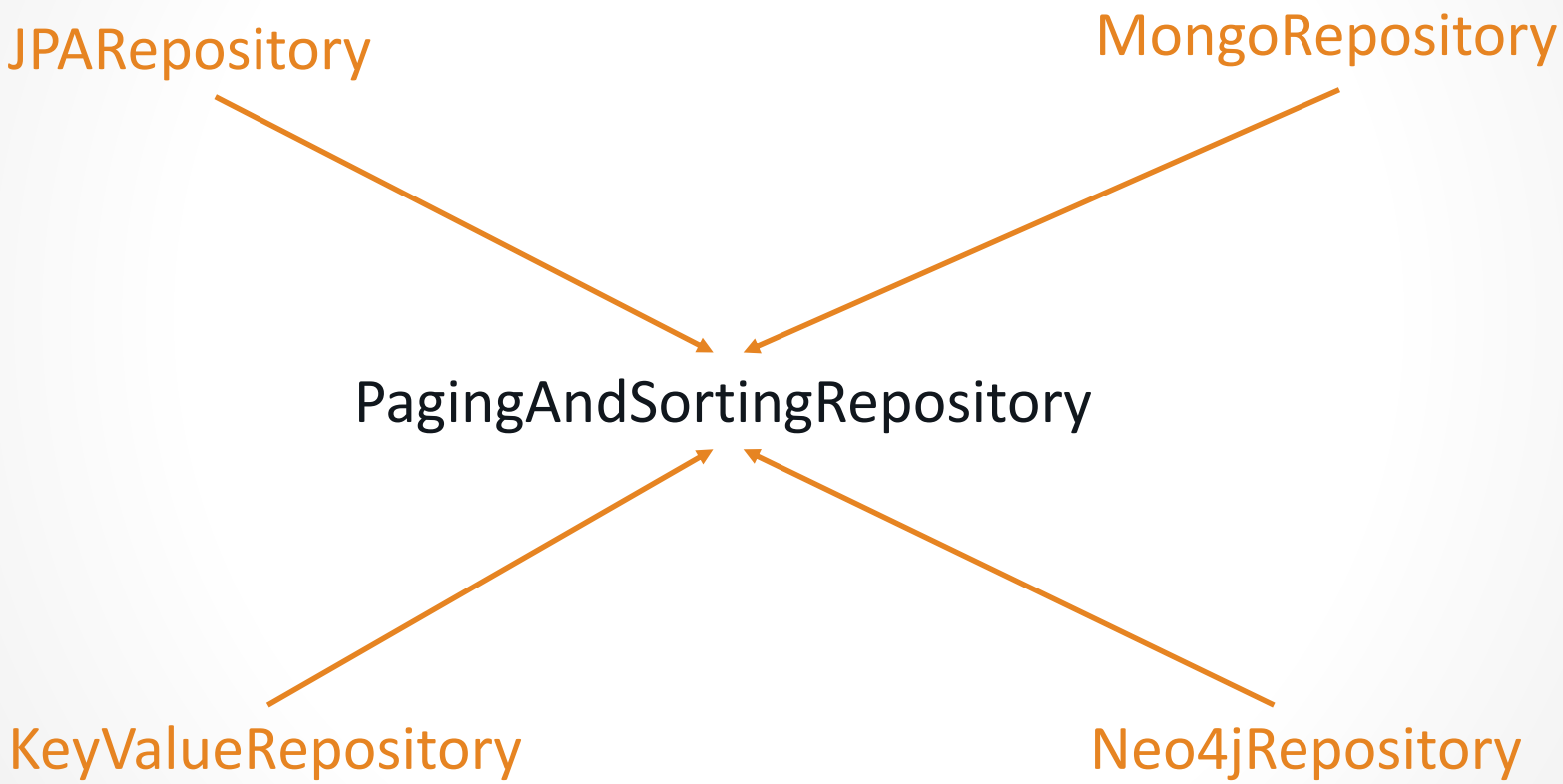
AOP proxy around EntityManagerFactory

```
public void delete(int id) {
    Product product = em.find(Product.class, id);
    if(product != null) {
        em.remove(product);
```

Spring Data JPA. Hierarchy



Spring Data. Repositories



Spring Data JPA. CRUD API


```
public interface ProductRepository extends
    JpaRepository<Product, Integer> {
```

```
@RequiredArgsConstructor
@Service
public class ProductService {

    private final ProductRepository productRepository;

    public void delete(int id) {
        productRepository.deleteById(id);
```

```
Product updated = productRepository.save(product);
```

```
productRepository.findById(id).ifPresentOrElse(
        activate(id), logError(id));
```

Spring Data JPA. CRUD API


```
public interface ProductRepository extends
    JpaRepository<Product, Integer> {

    @Override
    @Modifying
    @Query("UPDATE Product SET active=false WHERE id=:id")
    void deleteById(@Param("id") Integer id);
```

Spring Data JPA. Repositories

Project

Task

JPA

```
@PersistenceContext
private EntityManager em;
```

ProjectMember

ProjectConfig

ProjectHistory

Create a repository for each aggregated root

Spring Data JPA. Repositories


```
public interface OrderRepository extends
            CrudRepository<Order, Integer> {
```

```
public interface ProductRepository extends
    JpaRepository<Product, Integer> {
```

```
public interface CustomerRepository extends
            JpaRepository<Customer, Integer> {
```

save
delete
findById

Spring Data JPA. Repositories

```
public class PersistenceConfig {

    @PersistenceContext
    private EntityManager em;

    @Bean
    public SimpleJpaRepository<Product, Integer>
                                productRepository() {
        return new SimpleJpaRepository<>(Product.class, em);
    }
```

Only if you need basic JPARepository functionality

Case #3. Static queries

GET /products/1

JPA. Static queries


```
product = em.find(Product.class, product.getId());
```

SELECT * FROM Product WHERE id=?

Load entity from EntityManager/DB

```
TypedQuery<Product> query = em.createQuery(
        "from Product", Product.class);
List<Product> products = query.getResultList();
```

SELECT * FROM Product

```
TypedQuery<Product> query = em.createQuery(
        "FROM Product WHERE name=:name", Product.class)
        .setParameter("name", "Phone");
```

SELECT * FROM Product WHERE name=?

Spring Data JPA. Static queries


```
public interface ProductRepository extends
    CrudRepository<Product, Integer> {

    public List<Product> findByName(String name);

}
```

SELECT … FROM Product WHERE name= …

JPQL parameter

name value	Query
PC	FROM Product WHERE name='PC'
null	FROM Product WHERE name is null

```
Optional<Product> findByName(String name);
```

Spring Data JPA. Static queries

Field names

```
List<Product> findByNameAndPriceAndActiveTrue(String name,
        double price);

long countByPrice(double price);

boolean existsByName(String name);
```

SELECT COUNT(*) FROM Product WHERE price = …

SELECT id FROM Product WHERE price = …

Spring Data JPA. Static queries

Parameter name

```
@Query("from Product where name=:name")
public List<Product> findUsingName(@Param("name")
                String name);
```

```
public interface OrderRepository extends
            JpaRepository<Order, Integer> {
    int countByProduct_Id(int productId);
```

@ManyToOne(optional=false)

SELECT count(*) from Order where ….

@ManyToOne

SELECT count(*) from Order left outer join Product where ….

Spring Data JPA. Static queries


```
public interface ProductRepository extends
    JpaRepository<Product, Integer> {

    @Query("SELECT true FROM Product p WHERE p.id=:id")
    boolean exists(@Param("id") int id);
```

1

```
@Query("SELECT p FROM Product p WHERE p.id=:id")
```

2

```
@Query("SELECT p.id is null FROM Product p WHERE p.id=:id")
```

3

```
@Query("SELECT count(p) > 0 FROM Product p WHERE p.id=:id")
```

4

```
@Query("SELECT CASE WHEN COUNT(p) > 0 THEN true " +
        "ELSE false END FROM Product p WHERE p.id=:id")
```

5

Spring Data JPA. Asynchronous queries


```
public interface ProductRepository extends
    JpaRepository<Product, Integer> {

    @Async
    CompletableFuture<List<Product>> findByName(String name);
```

```
productRepository.findByName("Galaxy").
    whenComplete((result, ex) ->
    System.out.println(result));
```

Case #4. Pagination and sorting

GET /products?page_index=0&page_size=10&sort=name&dir=asc

JPA. Server-side pagination


```
TypedQuery<Product> query = em.createQuery(
        "FROM Product", Product.class);
query.setFirstResult(0);
query.setMaxResults(10);

List<Product> items = query.getResultList();
```

Starting index

Page size

Spring Data JPA. Pagination and sorting

```
@NoRepositoryBean
public interface PagingAndSortingRepository<T, ID> extends CrudRepository<
```

```
Iterable<T> findAll(Sort sort);
```

```
Page<T> findAll(Pageable pageable);
```

Pagination and sorting separated from query feature

```
Page<Product> page = productRepository
        .findAll(PageRequest.of(0, 5));
List<Product> products = page.getContent();
```

```
Page<Product> page = productRepository
        .findAll(PageRequest.of(0, 5, Sort.by(
                Direction.ASC, "name")));
List<Product> products = page.getContent();
```

Spring Data JPA. Pagination and sorting

```
public interface ProductRepository extends
    JpaRepository<Product, Integer> {

    List<Product> findByPrice(double price);
```

```
List<Product> findByPrice(double price, Sort sort);
```

```
List<Product> list = productRepository.findByPrice(200,
        Sort.by(Direction.ASC, "name"));
```

```
List<Product> list = productRepository.findByPrice(200,
        Sort.unsorted());
```

Spring Data JPA. Pagination and sorting

```
public interface ProductRepository extends
    JpaRepository<Product, Integer> {

 @Override
 @Query("SELECT p FROM Product p WHERE p.active=true")
 Page<Product> findAll(Pageable page);
```

```
@Query("SELECT p FROM Product p left join fetch p.orders " +
        "WHERE p.active=true")
Page<Product> findAll(Pageable page);
```

Caused by: org.hibernate.QueryException: query specified join fetching, but the owner of the fetched association was not present in the select list
[select count(p) FROM model.Product p left join fetch p.orders WHERE p.active=true]

Spring Data JPA. Pagination and sorting

```
@Override
@Query(value = "SELECT p FROM Product p left join fetch p.orders " +
        "WHERE p.active=true", countQuery = "SELECT COUNT(p) " +
        "FROM Product p WHERE p.active=true")
Page<Product> findAll(Pageable page);
```

Spring Data JPA. Pagination

PageImpl SliceImpl

Chunk

```
public List<T> getContent() {
    return Collections.unmodifiableList(content);
}
```

```
/**
 * Returns the page content as {@link List}.
 *
 * @return
 */
List<T> getContent();
```

Slice

Case #5. Projections (tuples)

JPA. Tuples


```
@RequiredArgsConstructor
@Getter
public class Tuple {
    private final int id;

    private final String name;

}
```

```
TypedQuery<Tuple> query = em.createQuery(
        "select new model.Tuple(id,name) "+
                "FROM Product", Tuple.class);

List<Tuple> items = query.getResultList();
```

Spring Data JPA. Projections


```
public interface OrderNumber {
    int getNumber();
}
```

Spring Data generate proxy for it

```
@Getter
@RequiredArgsConstructor
public class OrderDTO {
    private final int number;
}
```

```
public interface OrderRepository extends
            JpaRepository<Order, Integer> {

    List<OrderDTO> findDTOById(int id);

    List<OrderNumber> findAllById(int id);
```

Hibernate: select order0_.number as col_0_0_ from ORDERS order0_ where order0_.id=?

Spring Data JPA. Advanced projections


```
public interface OrderNumber {
    @Value("#{target.number + '#' + target.orderDate}")
    String info();

    int getNumber();

    LocalDateTime getOrderDate();
}
```

Case #7. Dynamic queries

GET /products?categoryId=1&status=in_stock

JPA. Named queries

Named queries are parsed/checked when context bootstraps

```
@Table
@Entity
@NamedQuery(name = Product.SELECT_QUERY, query = "FROM Product")
public class Product extends BaseEntity {

    public static final String SELECT_QUERY = "Product.selectAll";
```

JPA. Dynamic queries using JPQL


```java
@Service
public class ProductService {

    @PersistenceContext
    private EntityManager em;
```

```java
public List<Product> search(String name, Double price) {
    String jpql = "FROM Product";
    List<String> predicates = new ArrayList<>();
    if (name != null) {
        predicates.add("name=:name");
    }
    if (price != null) {
        predicates.add("price=:price");
    }
    String where = predicates.stream().collect(
            Collectors.joining(" AND "));
    if (!where.isEmpty()) {
        jpql += " WHERE " + where;
    }
    TypedQuery<Product> query = em.createQuery(jpql, Product.class);
    if(name != null) {
        query.setParameter("name", name);
    }
```

Compose JPQL

Initialize parameters

JPA. Dynamic queries and JPA Criteria


```
@Service
public class ProductService {

    @PersistenceContext
    private EntityManager em;

public List<Product> search(String name, Double price) {
    CriteriaBuilder cb = em.getCriteriaBuilder();

    CriteriaQuery<Product> cq = cb.createQuery(Product.class);
    Root<Product> root = cq.from(Product.class);
    cq.select(root);
    List<Predicate> predicates = new ArrayList<>();
    if (name != null) {
        predicates.add(cb.equal(root.get("name"), name));
    }
    if (price != null) {
        predicates.add(cb.equal(root.get("price"), price));
    }
    cq.where(cb.and(predicates.toArray(new Predicate[0])));

    return em.createQuery(cq).getResultList();
```

Spring Data JPA. Query by example

Run-time query (built dynamically based on input parameters)

```
Product product = new Product();
product.setName("phone1");
Example<Product> example = Example.of(product);

List<Product> items = productRepository.findAll(example);
```

where product0_.ID=0 and product0_.NAME=? and product0_.price=0.0

Match all non-null properties using AND condition

```
public class Product extends BaseEntity {

    private String name;

    private Double price;
```

Ignore NULL values

Spring Data JPA. Matchers


```
ExampleMatcher exampleMatcher = ExampleMatcher
        .matching()
        .withIgnorePaths("price")
        .withIncludeNullValues();
```

where product0_.NAME=?

```
ExampleMatcher exampleMatcher = ExampleMatcher
        .matchingAny();
```

Use OR to join non-null conditions

Spring Data JPA. Query by Example

```
Order order = new Order();
order.setAmount(20);
order.setProduct(productRepository.getOne(1));

Example<Order> example = Example.of(order);
List<Order> items = orderRepository.findAll(example);
```

Will include all product fields (not only id)

Spring Data JPA. Query by Example

```
Order order = new Order();
order.setAmount(20);

product = new Product();
product.setId(1);
order.setProduct(product);

Example<Order> example = Example.of(order);
List<Order> items = orderRepository.findAll(example);
```

from Orders o inner join Products p on p.id= o.product_id
where p.id=1 and o.amount=20

Spring Data JPA. Query by Example


```
product = new Product();
order = new Order();
order.setAmount(5);
order.setProduct(product);
product.setOrders(Set.of(order));

Example<Product> example = Example.of(product);
List<Product> items = productRepository.findAll(example);
```

FROM Product p

Spring Data JPA. Specification


```
public interface ProductRepository extends
    JpaRepository<Product, Integer>,
    JpaSpecificationExecutor<Product> {
```

All the checks are made at run-time

```
private final ProductRepository productRepository;

public List<Product> search(String name, Double price) {
    Specification<Product> spec = (root, cq, cb) -> {

        List<Predicate> predicates = new ArrayList<>();
        if (name != null) {
            predicates.add(cb.equal(root.get("name"), name));
        }
        if (price != null) {
            predicates.add(cb.equal(root.get("price"), price));
        }
        return cb.and(predicates.toArray(new Predicate[0]));
    };
    return productRepository.findAll(spec);
```

Case #6. Audit

Audit. JPA


```java
@Column(updatable = false)
private LocalDateTime created;

@Column(insertable = false)
private LocalDateTime modified;
```

```java
@PrePersist
public void onPersist() {
    created = LocalDateTime.now();
}

@PreUpdate
public void onUpdate() {
    modified = LocalDateTime.now();
}
```

```java
@PreRemove
public void onRemove() {

}
```

Hibernate annotations

```java
@CreationTimestamp
private LocalDateTime created;

@UpdateTimestamp
private LocalDateTime modified;
```

Audit. Spring Data JPA


```
@CreatedDate
private LocalDateTime created;

@LastModifiedDate
private LocalDateTime modified;
```

Spring Data Commons annotations

```
@Configuration
@EnableJpaAuditing
public class PersistenceConfig {
```

Could not configure Spring Data JPA auditing-feature because spring-aspects.jar is not on the classpath!

Add **spring-aspects** dependency

```
@EntityListeners(AuditingEntityListener.class)
public abstract class BaseEntity {
```

Case #8. Transactions

Plain JPA. Transactions


```
EntityManager em = null;
try {
    em = emf.createEntityManager();
    em.getTransaction().begin();

    em.getTransaction().commit();
} catch (Exception ex) {
    ex.printStackTrace();
    if (em != null && em.getTransaction().isActive()) {
        em.getTransaction().rollback();
    }
} finally {
    em.close();
    emf.close();
}
```

Any **modification** operation should be wrapper into a transaction

Spring JPA. Declarative transactions

```
@Service
@Transactional
public class ProductService {

    @PersistenceContext
    private EntityManager em;
```

Wrap each call into single transaction using AOP

```
@Service
@Transactional(readOnly = true, propagation = Propagation.NESTED,
        rollbackFor = IOException.class)
public class ProductService {
```

Spring Data JPA. Transactions


```
@Repository
@Transactional(readOnly = true)
public class SimpleJpaRepository<T, ID>
```

```
@Transactional
@Override
public <S extends T> S save(S entity) {
```

Write-concerned transaction

```
@Query("UPDATE Product p SET p.active = false where p.id=:id")
@Modifying
@Transactional(propagation = Propagation.REQUIRES_NEW)
void deactive(@Param("id") int productId);
```

@Transactional is required for custom update/delete query

Case #9. Caching

Caching. JPA & Hibernate

Pure **Hibernate** feature

```
@Entity
@Table(name="PRODUCTS")
@Cache(usage=CacheConcurrencyStrategy.READ_WRITE)
public class Product extends BaseEntity {
    private String name;
```

```
hibernate.cache.use_query_cache=true
```

```
TypedQuery<Product> query = em.createQuery(
        "FROM Product WHERE id =:id", Product.class)
        .setParameter("id", productId);

query.setHint(QueryHints.CACHEABLE, true);
product = query.getSingleResult();
```

Cache entity identifiers

Hibernate hint

Caching. Spring Data JPA


```
hibernate.cache.use_query_cache=true
```

Not cacheable by default

```
@Query("from Product where name=:name")
public List<Product> findUsingName(@Param("name")
                String name);
```

Based on Spring Cache
and requires cache provider

```
@Query("from Product where name=:name")
@Cacheable(cacheNames = "products")
List<Product> findUsingName(@Param("name") String name);
```

```
@SpringBootApplication
@EnableCaching
public class MainApplication {
```

Case #10. Performance & optimization

Spring Data JPA. Performance

- Spring Data JPA is an abstraction layer on top of JPA
- Repositories discovery
- Static queries analysis and execution

Spring Data JPA. Optimization


```
@SpringBootApplication
@EnableJpaRepositories(bootstrapMode = BootstrapMode.DEFERRED)
public class MainApplication {
```

```
@RequiredArgsConstructor
class DeferredRepositoryInitializationListener implements ApplicationListener

    private final @NonNull ListableBeanFactory beanFactory;
```

```
@Override
public void onApplicationEvent(ContextRefreshedEvent event) {

    LOG.info("Triggering deferred initialization of Spring Data repositor

    beanFactory.getBeansOfType(Repository.class);
```

Finished Spring Data repository scanning in **672ms.**
Found **43 JPA** repository interfaces.

- Sergiy Morenets, 2020

Spring Data vs JPA. Benchmarks

Find entity by parameter

1. JPQL and named query
2. JPA criteria
3. Spring Data JPA with query methods
4. Spring Data JPA with custom JPQL
5. Spring Data JPA with query by example

Benchmarks. JMH

- ✓ **JMH** is micro benchmarking framework
- ✓ Developed by **Oracle** engineers
- ✓ First release in 2013
- ✓ Requires build tool (Maven, **Gradle**)
- ✓ Can measure throughput or average time
- ✓ Includes **warm-up** period
- ~~✓ Part of Java 9~~

Benchmarks. Environment

- **JMH** 1.25
- **Hibernate** 5.4.20
- **Maven** 3.6
- **JDK** 15.0.1
- Intel Core **i9,** 8 cores, 32 GB

Benchmarks

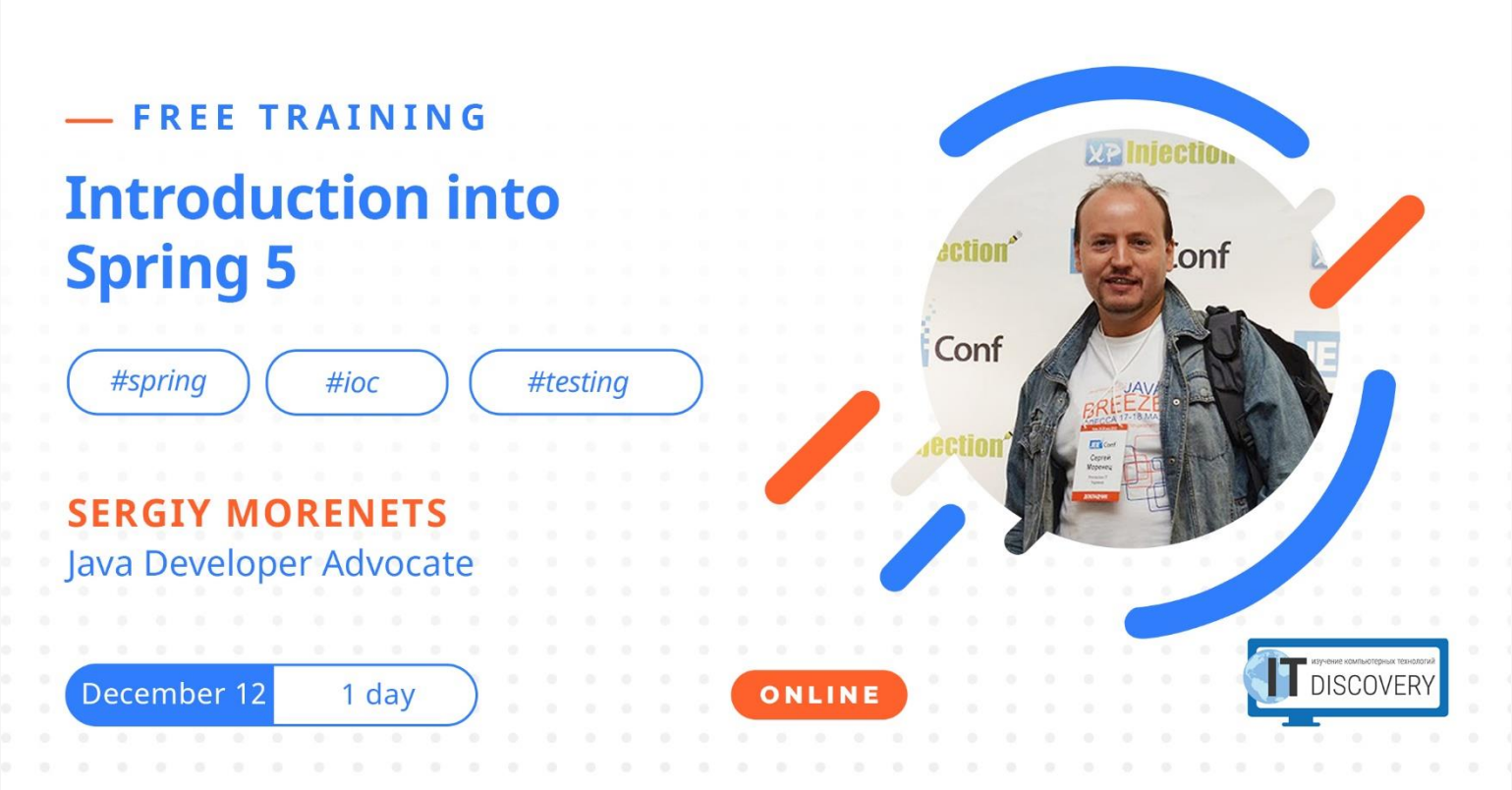

```
@State(Scope.Thread)
@BenchmarkMode(Mode.AverageTime)
@Warmup(iterations = 10)
@Measurement(iterations = 10)
public class Benchmarking {

    private EntityManager em;
```

```
@Benchmark
public List<Product> jpql() {
    return em.createNamedQuery(Product.FIND_QUERY, Product.class)
            .setParameter("name", "phone")
            .getResultList();
}
```

Benchmarks. Results

Case	Time(ns)	Error(ns)
JPA (JPQL)	**4280**	15
JPA Criteria API	5103	109
Spring Data Query	11609	19
Spring Data (JPQL)	10466	20
Spring Data Query by Example	17887	48

- https://it-simulator.com/
- Sergiy Morenets, sergey.morenets@gmail.com



- Sergiy Morenets, 2020