Pengenalan Spring Data JPA

Pengenalan Spring Data JPA

- Spring Data JPA adalah fitur di Spring yang digunakan untuk mempermudah kita membuat aplikasi menggunakan JPA
- Dengan menggunakan Spring Data JPA, kita tidak perlu membuat banyak hal secara manual dilakukan seperti jika menggunakan JPA secara manual
- https://spring.io/projects/spring-data-jpa

Spring Data

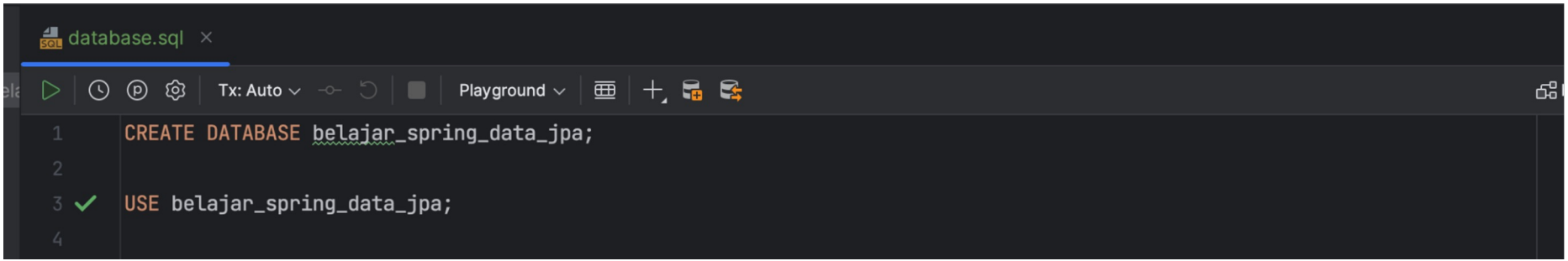
- Spring Data JPA sendiri merupakan bagian dari Spring Data Project
- Spring Data Project adalah project di Spring yang digunakan untuk mempermudah kita melakukan manipulasi data di database
- Konsep yang digunakan di Spring Data hampir sama di semua tipe project, oleh karena itu jika nanti kita sudah mengerti tentang Spring Data JPA, ketika akan menggunakan fitur Spring Data lainnya, maka tidak akan kesulitan, seperti Spring Data MongoDB, Spring Data Elasticsearch, Spring Data Redis, dan lain-lain
- https://spring.io/projects/spring-data

Membuat Project

Membuat Project

- https://start.spring.io/
- Spring Web MVC
- Spring Data JPA
- Lombok
- MySQL
- Validation

Membuat Database



Data Source

DataSource

- Salah satu keuntungan menggunakan Spring Data JPA dan Spring Boot adalah, semua upacara yang biasa kita lakukan ketika menggunakan JPA, sudah dilakukan oleh Spring Boot
- Jadi kita tidak perlu membuat DataSource secara manual, karena sudah otomatis dibuat oleh Spring Boot
- https://github.com/spring-projects/spring-boot/blob/main/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/jdbc/DataSourceAutoConfiguration.java
- Untuk mengubah konfigurasi DataSource, kita cukup menggunakan application properties saja
- Kita bisa lihat semua konfigurasinya dengan prefix spring.datasource.*
- https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html#appendix.application-properties.data

Kode : Konfigurasi DataSource

application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=
spring.datasource.url=jdbc:mysql://localhost:3306/belajar_spring_data_jpa
spring.datasource.type=com.zaxxer.hikari.HikariDataSource
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=10
```

JPA Configuration

Konfigurasi JPA

- Untuk melakukan konfigurasi JPA, kita juga tidak perlu melakukannya secara manual lagi di file persistence.xml
- Secara otomatis JPA akan menggunakan DataSource di Spring, dan jika kita butuh mengubah konfigurasi, kita bisa menggunakan properties dengan prefix spring.jpa.*
- https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html

Kode : Konfigurasi JPA

application.properties

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.username=root
spring.datasource.password=
spring.datasource.url=jdbc:mysql://localhost:3306/belajar_spring_data_jpa
spring.datasource.type=com.zaxxer.hikari.HikariDataSource
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=10

spring.jpa.properties.hibernate.show_sql=true
spring.jpa.properties.hibernate.format_sql=true

```

Entity Manager Factory

Entity Manager Factory

- Selain DataSource, Spring Boot juga secara otomatis membuatkan bean EntityMangerFactory, sehingga kita tidak perlu membuatnya secara manual
- Itu semua secara otomatis dibuat oleh Spring Boot
- https://github.com/spring-projects/spring-boot/blob/main/spring-boot-project/spring-boot-autoconfigure/src/main/java/org/springframework/boot/autoconfigure/orm/jpa/HibernateJpaAutoConfiguration.java

Kode : Entity Manager Factory

```java
@SpringBootTest
public class EntityManagerTest {

    @Autowired
    private EntityManagerFactory entityManagerFactory;

    new *
    @Test
    void testEntityManagerFactory() {
        assertNotNull(entityManagerFactory);

        EntityManager entityManager = entityManagerFactory.createEntityManager();
        assertNotNull(entityManager);

        entityManager.close();
    }
```

Repository

Tanpa Entity Manager

- Saat kita menggunakan Spring Data JPA, kita akan jarang sekali membuat Entity Manager lagi
- Bahkan mungkin jarang menggunakan Entity Manager Factory
- Spring Data membawa konsep Repository (diambil dari buku Domain Driven Design)
- Dimana Repository merupakan layer yang digunakan untuk mengelola data (contohnya di database)

Repository

- Setiap Entity yang kita buat di JPA, maka kita biasanya akan buatkan Repository nya
- Repository berbentuk Interface, yang secara otomatis diimplementasikan oleh Spring menggunakan AOP
- Untuk membuat Repository, kita cukup membuat interface turunan dari JPARepository<T, ID>
- https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaRepository.html
- Dan kita juga bisa tambahkan annotation @Repository (walaupun tidak wajib)
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/stereotype/Repository.html

Kode : Create Table Category

```sql
CREATE TABLE categories
(
    id   BIGINT       NOT NULL AUTO_INCREMENT,
    name VARCHAR(100) NOT NULL,
    PRIMARY KEY (id)
) ENGINE InnoDB;
```

Kode : Category Entity

```java
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "categories")
public class Category {

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

    private String name;
}
```

Kode : Category Repository

```java
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;
import programmerzamannow.jpa.entity.Category;

no usages  new *
@Repository
public interface CategoryRepository extends JpaRepository<Category, Long> {

}
  You, 2 minutes ago • Uncommitted changes
```

Crud Repository

Category Repository

- JpaRepository adalah turunan dari interface CrudRepository dan ListCrudRepository, dimana di interface tersebut banyak method yang bisa digunakan untuk melakukan operasi CRUD
- Kita tidak perlu lagi menggunakan Entity Manager untuk melakukan operasi CRUD, cukup gunakan JpaRepository
- Ada yang perlu diperhatikan di JpaRepository, method untuk CREATE dan UPDATE digabung dalam satu method save(), yang artinya method save() adalah CREATE or UPDATE
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/CrudRepository.html
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/ListCrudRepository.html

Kode : Test Create Category Repository

Navigate to Application Context

```java
class CategoryRepositoryTest {

    @Autowired
    private CategoryRepository categoryRepository;

    new *
    @Test
    void testCreate() {
        Category category = new Category();
        category.setName("GADGET");

        categoryRepository.save(category);

        assertNotNull(category.getId());
    }
```

Kode : Test Update Category Repository

```
new *
@Test
void testUpdate() {
    Category category = categoryRepository.findById(1L).orElse(null);
    assertNotNull(category);

    category.setName("GADGET MURAH");
    categoryRepository.save(category);

    category = categoryRepository.findById(1L).orElse(null);
    assertNotNull(category);
    assertEquals("GADGET MURAH", category.getName());
}
}
```

Declarative Transaction

Declarative Transaction

- Saat kita menggunakan JPA secara manual, kita harus melakukan management transaction secara manual menggunakan EntityManager
- Spring menyediakan fitur Declarative Transaction, yaitu management transaction secara declarative, yaitu dengan menggunakan annotation @Transactional
- Annotation ini secara otomatis dibaca oleh Spring AOP, dan akan menjalankan transaction secara otomatis ketika memanggil method yang terdapat annotation @Transactional nya
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/annotation/Transactional.html

Yang Perlu Diperhatikan

- Saat membuat method dengan annotation @Transactional, karena dia dibungkus oleh Spring AOP, jadi untuk menjalankannya, kita harus memanggil method tersebut dari luar object
- Misal kita memiliki CategoryService.create() dengan annotation @Transactional, jika kita panggil dari CategoryController, maka Spring AOP akan berjalan, namun jika dipanggil di CategoryService.test() misalnya, maka Spring AOP tidak akan berjalan

Kode : Category Service

```java
@Service
public class CategoryService {

    @Autowired
    private CategoryRepository categoryRepository;

    1 usage  new *
    @Transactional
    public void create() {
        for (int i = 0; i < 5; i++) {
            Category category = new Category();
            category.setName("Category " + i);
            categoryRepository.save(category);
        }
        throw new RuntimeException("Ups rollback please");
    }

    new *
    public void test() {
        create();
    }
```

Kode : Test Category Service

```
new *
@Test
void success() {
    assertThrows(RuntimeException.class, () -> {
        categoryService.create();
    });
}

new *
@Test
void failed() {
    assertThrows(RuntimeException.class, () -> {
        categoryService.test();
    });
}
```

Transaction Propagation

- Saat kita membuat method dengan annotation @Transactional, kita mungkin didalamnya memanggil method @Transactional lainnya
- Pada kasus seperti itu, ada baiknya kita mengerti tentang attribute propagation pada @Transactional
- Kita bisa memilih nilai apa yang ingin kita gunakan
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/annotation/Propagation.html

Programmatic Transaction

Programmatic Transaction

- Fitur Declarative Transaction sangat mudah untuk digunakan, karena hanya butuh menggunakan annotation
- Namun pada beberapa kasus, misal kode yang kita buat butuh jalan secara async misal nya, maka Declarative Transaction tidak akan berjalan, mau tidak mau biasanya kita akan melakukan manual transaction management lagi
- Kita bisa gunakan cara lama menggunakan Entity Manager, atau kita bisa menggunakan fitur Spring untuk melakukan management transaction secara manual
- Ada beberapa cara untuk melakukan programmatic transaction di Spring

Transaction Operations

- Pada kasus yang sederhana, kita bisa menggunakan TransactionOperations
- Kita bisa menggunakan bean TransactionOperations yang sudah secara otomatis dibuat oleh Spring Boot
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/support/TransactionOperations.html

Kode : Transaction Operations

```java
@Autowired
private TransactionOperations transactionOperations;

2 usages  new *
public void error() {
    throw new RuntimeException("Ups");
}

no usages  new *
public void createCategories() {
    transactionOperations.executeWithoutResult(transactionStatus -> {
        for (int i = 0; i < 5; i++) {
            Category category = new Category();
            category.setName("Category " + i);   You, Moments ago • Uncommitted changes
            categoryRepository.save(category);
        }
        error();
    });
```

Platform Transaction Manager

- Jika kita butuh melakukan management transaction secara low level, maka sebenarnya kita bisa menggunakan Entity Manager, namun hal itu tidak disarankan
- Kita bisa menggunakan Platform Transaction Manager yang sudah disediakan oleh Spring
- https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/transaction/PlatformTransactionManager.html
- Penggunaan ini sangat manual, sehingga kita bisa atur semuanya secara manual

Kode : Platform Transaction Manager

```java
@Autowired
private PlatformTransactionManager transactionManager;

1 usage  new *
public void manual() {
    DefaultTransactionDefinition definition = new DefaultTransactionDefinition();
    definition.setTimeout(10);
    definition.setPropagationBehavior(TransactionDefinition.PROPAGATION_REQUIRED);

    TransactionStatus transaction = transactionManager.getTransaction(definition);
    try {
        for (int i = 0; i < 5; i++) {
            Category category = new Category();
            category.setName("Category Manual " + i);
            categoryRepository.save(category);
        }
        error();
        transactionManager.commit(transaction);
    } catch (Throwable throwable) {
        transactionManager.rollback(transaction);
        throw throwable;
```

Query Method

Query Method

- Saat kita menggunakan EntityManager, kita bisa membuat query menggunakan JPA QL, namun bagaimana jika menggunakan Repository?
- Spring Data menyediakan fitur Query Method, yaitu membuat query menggunakan nama method secara otomatis
- Spring Data akan melakukan penerjemahan secara otomatis dari nama method menjadi JPA QL

Format Query Method

- Untuk melakukan query yang mengembalikan data lebih dari satu, kita bisa gunakan prefix findAll...
- Untuk melakukan query yang mengembalikan data pertama, kita bisa gunakan prefix findFirst...
- Selanjutnya diikuti dengan kata By dan diikuti dengan operator query nya
- Untuk operator query yang didukung, kita bisa lihat di halaman ini
- https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods.query-creation

Kode : Category Repository

```
10  @Repository
11  public interface CategoryRepository extends JpaRepository<Category, Long> {
12
        no usages  new *
13      Optional<Category> findFirstByNameEquals(String name);
14
        no usages  new *
15      List<Category> findAllByNameLike(String name);
16
17  }
18      You, Moments ago • Uncommitted changes
```

Kode : Test Category Repository

```
    @Test
    void testQueryMethod() {
        Category category = categoryRepository.findFirstByNameEquals("GADGET MURAH").orElse(null);
        assertNotNull(category);
        assertEquals("GADGET MURAH", category.getName());

        List<Category> categories = categoryRepository.findAllByNameLike("%GADGET%");
        assertEquals(1, categories.size());
        assertEquals("GADGET MURAH", categories.get(0).getName());
    }
```

Query Relation

Query Relation

- Saat kita belajar JPA, kita bisa melakukan query ke relasi Entity atau Embedded field secara otomatis menggunakan tanda . (titik)
- Di Spring Data Repository, kita bisa gunakan _ (garis bawah) untuk menyebutkan bahwa itu adalah tanda . (titik) nya
- Misal ProductRepository.findAllByCategory_Name(String)

Kode : Create table Product

```
CREATE TABLE products
(
    id          BIGINT       NOT NULL AUTO_INCREMENT,
    name        VARCHAR(100) NOT NULL,
    price       BIGINT       NOT NULL,
    category_id BIGINT       NOT NULL,
    primary key (id),
    foreign key fk_products_categories (category_id) REFERENCES categories (id)
) ENGINE InnoDB;
```

Kode : Product Entity

```java
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "products")
public class Product {

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

    private String name;

    private Long price;

    @ManyToOne
    @JoinColumn(name = "category_id", referencedColumnName = "id")
    private Category category;
}
```

```java
@Getter
@Setter
@AllArgsConstructor
@NoArgsConstructor
@Entity
@Table(name = "categories")
public class Category {

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

    private String name;

    @OneToMany(mappedBy = "category")
    private List<Product> products;
}
```

Kode : Product Repository

```
no usages  new *
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    no usages  new *
    List<Product> findAllByCategory_Name(String name);

}
   You, Moments ago • Uncommitted changes
```

Kode : Test Product Repository

```java
@Test
void createProduct() {
    Category category = categoryRepository.findById(1L).orElse(null);
    assertNotNull(category);

    {
        Product product = new Product();
        product.setName("Apple iPhone 14 Pro Max");
        product.setPrice(25_000_000L);
        product.setCategory(category);
        productRepository.save(product);
    }

    {
        Product product = new Product();
        product.setName("Apple iPhone 13 Pro Max");
        product.setPrice(18_000_000L);
        product.setCategory(category);
        productRepository.save(product);
    }
}
```

```java
@Test
void findProducts() {
    List<Product> products = productRepository
            .findAllByCategory_Name("GADGET MURAH");

    assertEquals(2, products.size());
    assertEquals("Apple iPhone 14 Pro Max", products.get(0).getName());
    assertEquals("Apple iPhone 13 Pro Max", products.get(1).getName());
}
```

Sorting

Sorting

- Spring Data Repository juga memiliki fitur untuk melakukan Sorting, caranya kita bisa tambahkan parameter Sort pada posisi parameter terakhir
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Sort.html

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    1 usage  new *
    List<Product> findAllByCategory_Name(String name);

    no usages  new *
    List<Product> findAllByCategory_Name(String name, Sort sort);

}
```

You, Moments ago • Uncommitted changes

Kode : Test Product Repository

```java
@Test
void findProductsSort() {
    Sort sort = Sort.by(Sort.Order.desc("id"));
    List<Product> products = productRepository.findAllByCategory_Name("GADGET MURAH", sort);

    assertEquals(2, products.size());
    assertEquals("Apple iPhone 13 Pro Max", products.get(0).getName());
    assertEquals("Apple iPhone 14 Pro Max", products.get(1).getName());
}
```

Paging

Paging

- Selain Sort, Spring Data Repository juga mendukung paging seperti di EntityManager
- Caranya kita bisa tambahkan parameter Pageable di posisi terakhir parameter
- Pageable adalah sebuah interface, biasanya kita akan menggunakan PageRequest sebagai class implementasinya
- Dan jika sudah menggunakan Pageable, kita tidak perlu lagi menggunakan Sort, karena sudah bisa dihandle oleh Pageable
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/PageRequest.html

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    1 usage  new *
    List<Product> findAllByCategory_Name(String name);

    1 usage  new *
    List<Product> findAllByCategory_Name(String name, Sort sort);

    2 usages  new *
    List<Product> findAllByCategory_Name(String name, Pageable pageable);

}
```

Kode : Test Product Repository

```
@Test
void testFindProductsWithPageable() {
    // page 0
    Pageable pageable = PageRequest.of(0, 1, Sort.by(Sort.Order.desc("id")));
    List<Product> products = productRepository.findAllByCategory_Name("GADGET MURAH", pageable);

    assertEquals(1, products.size());
    assertEquals("Apple iPhone 13 Pro Max", products.get(0).getName());

    // page 1
    pageable = PageRequest.of(1, 1, Sort.by(Sort.Order.desc("id")));
    products = productRepository.findAllByCategory_Name("GADGET MURAH", pageable);

    assertEquals(1, products.size());
    assertEquals("Apple iPhone 14 Pro Max", products.get(0).getName());
}
```

Page Result

Page Result

- Saat kita menggunakan Paging, kadang kita ingin tahu seperti jumlah total data hasil query, dan juga total page nya
- Hal ini biasanya kita akan lakukan dengan cara manual dengan cara menghitung count dari hasil total hasil query tanpa paging
- Untungnya, Spring Data JPA menyediakan return value berupa Page<T>, dimana secara otomatis akan diambil informasi total data dan total page nya
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Page.html

Kode : Product Repository

```
1 usage  new *
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    1 usage  new *
    List<Product> findAllByCategory_Name(String name);

    1 usage  new *
    List<Product> findAllByCategory_Name(String name, Sort sort);

    2 usages  new *
    Page<Product> findAllByCategory_Name(String name, Pageable pageable);

}
```

Kode : Test Product Repository

```
    // page 0
    Pageable pageable = PageRequest.of(0, 1, Sort.by(Sort.Order.desc("id")));
    Page<Product> products = productRepository.findAllByCategory_Name("GADGET MURAH", pageable);

    assertEquals(1, products.getContent().size());
    assertEquals(0, products.getNumber());
    assertEquals(2, products.getTotalElements());
    assertEquals(2, products.getTotalPages());
    assertEquals("Apple iPhone 13 Pro Max", products.getContent().get(0).getName());

    // page 1
    pageable = PageRequest.of(1, 1, Sort.by(Sort.Order.desc("id")));
    products = productRepository.findAllByCategory_Name("GADGET MURAH", pageable);

    assertEquals(1, products.getContent().size());
    assertEquals(1, products.getNumber());
    assertEquals(2, products.getTotalElements());
    assertEquals(2, products.getTotalPages());
    assertEquals("Apple iPhone 14 Pro Max", products.getContent().get(0).getName());
}
```

Count Query Method

Count Query Method

- JPA Repository juga bisa digunakan untuk membuat count query method
- Cukup gunakan prefix method countBy...
- Selebihnya kita bisa membuat format seperti Query Method biasanya

Kode : Product Repository

```
1 usage  new *
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    no usages  new *
    Long countByCategory_Name(String name);
```

Kode : Test Product Repository

```java
@Test
void testCount() {
    Long count = productRepository.count();
    assertEquals(2L, count);

    count = productRepository.countByCategory_Name("GADGET MURAH");
    assertEquals(2L, count);
}
```

Exists Query Method

Exist Query Method

- Selain Count, kita juga bisa membuat Exists method di Query Method
- Method ini sebenarnya sederhana, return value nya adalah boolean, untuk mengecek apakah ada data sesuai dengan Query Method atau tidak
- Untuk membuatnya kita bisa gunakan prefix existsBy...

Kode : Product Category

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    no usages  new *
    boolean existsByName(String name);
```

Kode : Test Product Category

```
    @Test
    void testExists() {
        boolean exists = productRepository.existsByName("Apple iPhone 14 Pro Max");
        assertTrue(exists);

        // test not exists
        exists = productRepository.existsByName("Apple iPhone 14 Pro Max 2");
        assertFalse(exists);
    }
}
```

Delete Query Method

Delete Query Method

- Kita juga bisa membuat delete Query Method dengan prefix deleteBy
- Untuk delete, kita bisa return int sebagai penanda jumlah record yang berhasil di hapus
- Untuk membuat delete query method, kita bisa gunakan prefix deleteBy...

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    no usages  new *
    int deleteByName(String name);
```

Kode : Test Product Repository

```java
@Test
void testDelete() {
    transactionOperations.executeWithoutResult(transactionStatus -> {
        Category category = categoryRepository.findById(1L).orElse(null);
        assertNotNull(category);

        Product product = new Product();
        product.setName("Samsung Galaxy S9");
        product.setPrice(10_000_000L);
        product.setCategory(category);
        productRepository.save(product);

        int delete = productRepository.deleteByName("Samsung Galaxy S9");
        assertEquals(1, delete);

        // test not exists
        delete = productRepository.deleteByName("Samsung Galaxy S9");
        assertEquals(0, delete);
    });
```

Repository Transaction

Repository Transaction

- Secara default, saat kita membuat Repository interface, Spring akan membuat sebagai instance turunan dari SimpleJpaRepository
- Oleh karena itu, saat kita melakukan CRUD, kita tidak perlu melakukan didalam Transaction, hal ini karena sudah ditambahkan annotation di class SimpleJpaRepository
- Class SimpleJpaRepository terdapat annontatio @Transactional(readOnly=true), oleh karena itu saat kita buat Query Method di Repository, maka secara default akan menjalankan transaction read only
- https://docs.spring.io/spring-data/data-jpa/docs/current/api/org/springframework/data/jpa/repository/support/SimpleJpaRepository.html

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    2 usages  new *
    @Transactional
    int deleteByName(String name);
```

Kode : Test Product Repository

```
void testDelete() {
    Category category = categoryRepository.findById(1L).orElse(null);
    assertNotNull(category);

    Product product = new Product();
    product.setName("Samsung Galaxy S9");
    product.setPrice(10_000_000L);   You, Moments ago • Uncommitted changes
    product.setCategory(category);
    productRepository.save(product);

    int delete = productRepository.deleteByName("Samsung Galaxy S9");
    assertEquals(1, delete);

    // test not exists
    delete = productRepository.deleteByName("Samsung Galaxy S9");
    assertEquals(0, delete);
}
```

Named Query

Named Query

- Saat kita menggunakan JPA, kita sering sekali menggunakan Named Query
- Lantas bagaimana jika kita menggunakan Spring Data JPA Repository?
- Untuk menggunakan Named Query di Repository, kita cukup buat nama method sesuai degan nama Named Query, misal jika kita memiliki Named Query dengan nama Product.searchProductUsingName, maka kita bisa membuat method ProductRepository.searchProductUsingName()
- Secara otomatis itu akan menggunakan Named Query tersebut

Kode : Product Entity

```
@Entity
@Table(name = "products")
@NamedQueries({
        @NamedQuery(name = "Product.searchProductUsingName",
                query = "SELECT p FROM Product p WHERE p.name = :name"),
})
public class Product {

```

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    no usages  new *
    List<Product> searchProductUsingName(@Param("name") String name);
```

Kode : Test Product Repository

```java
    @Test
    void searchProduct() {
        List<Product> products = productRepository.searchProductUsingName("Apple iPhone 14 Pro Max");

        assertEquals(1, products.size());
        assertEquals("Apple iPhone 14 Pro Max", products.get(0).getName());
    }
}
```

Sorting dan Paging

- Named Query di Repository tidak mendukung Sort
- Namun mendukung Pageable (tanpa Sort), oleh karena itu kita harus menambahkan Sorting secara manual di Named Query nya

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    1 usage  new *
    List<Product> searchProductUsingName(@Param("name") String name, Pageable pageable);
```

```java
    @Test
    void searchProduct() {
        Pageable pageable = PageRequest.of(0, 1);
        List<Product> products = productRepository.searchProductUsingName("Apple iPhone 14 Pro Max", pageable);

        assertEquals(1, products.size());
        assertEquals("Apple iPhone 14 Pro Max", products.get(0).getName());
    }
```

Query Annotation

Query Annotation

- Query Method cocok untuk kasus membuat jenis query yang tidak terlalu kompleks. Saat query terlalu kompleks dan parameter banyak, maka nama method bisa terlalu panjang jika menggunakan Query Method
- Untungnya Spring Data JPA menyediakan membuat query menggunakan annotation Query, dimana kita bisa buat JPA QL atau Native Query
- https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/Query.html

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    no usages  new *
    @Query(value = "select p from Product p where p.name like :name or p.category.name like :name")
    List<Product> searchProduct(@Param("name") String name);
```

Kode : Test Product Repository

```java
@Test
void searchProductLike() {
    List<Product> products = productRepository.searchProduct("%iPhone%");
    assertEquals(2, products.size());

    products = productRepository.searchProduct("%GADGET%");
    assertEquals(2, products.size());
}
```

Sort dan Paging

- Query Annotation mendukung Sort dan Paging
- Jadi kita bisa menggunakan parameter Sort atau Pageable pada Query Annotation

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    2 usages  new *
    @Query(value = "select p from Product p where p.name like :name or p.category.name like :name")
    List<Product> searchProduct(@Param("name") String name, Pageable pageable);
```

```java
        Pageable pageable = PageRequest.of(0, 1, Sort.by(Sort.Order.desc("id")));

        List<Product> products = productRepository.searchProduct("%iPhone%", pageable);
        assertEquals(1, products.size());

        products = productRepository.searchProduct("%GADGET%", pageable);
        assertEquals(1, products.size());
    }
```

Page Result

- Sebelumnya kita sempat bahas tentang Page Result ketika menggunakan Paging
- Pada kasus jika kita ingin return dari Query Method nya adalah Page<T>, bukan List<T>, maka kita harus memberitahu Spring Data JPA bagaimana cara melakukan count query nya
- Kita bisa tambahkan query count nya pada attribute countQuery di Query Annotation

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    2 usages  new *
    @Query(
            value = "select p from Product p where p.name like :name or p.category.name like :name",
            countQuery = "select count(p) from Product p where p.name like :name or p.category.name like :name"
    )
    Page<Product> searchProduct(@Param("name") String name, Pageable pageable);
```

```java
        Pageable pageable = PageRequest.of(0, 1, Sort.by(Sort.Order.desc("id")));

        Page<Product> products = productRepository.searchProduct("%iPhone%", pageable);
        assertEquals(1, products.getContent().size());
```

Modifying

Modifying

- Query Annotation juga bisa digunakan untuk membuat JPQ QL atau Native Query untuk perintah Update atau Delete, caranya kita perlu menambahkan annotation @Modifying untuk memberitahu bahwa ini bukan Query Select
- https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/Modifying.html

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    no usages  new *
    @Modifying
    @Query("delete from Product p where p.name = :name")
    int deleteProductUsingName(@Param("name") String name);

    no usages  new *
    @Modifying
    @Query("update Product p set p.price = 0 where p.id = :id")
    int updateProductPriceToZero(@Param("id") Long id);
```

Kode : Test Product Repository

```
@Test
void modifying() {
    transactionOperations.executeWithoutResult(transactionStatus -> {
        int total = productRepository.deleteProductUsingName("Wrong");
        assertEquals(0, total);

        total = productRepository.updateProductPriceToZero(1L);
        assertEquals(1, total);

        Product product = productRepository.findById(1L).orElse(null);
        assertNotNull(product);
        assertEquals(0L, product.getPrice());
    });
}
```

Stream

Stream

- Saat kita menggunakan List<T> dan Query Method findAll..., maka secara otomatis seluruh data hasil dari database akan di load ke memory
- Pada kasus data yang sangat banyak, hal ini sangat berbahaya karena bisa terjadi error OutOfMemory
- Spring Data JPA bisa menggunakan fitur database cursor, untuk mengambil data sedikit demi sedikit ketika diperlukan menggunakan Java Stream
- Kita bisa membuat Query Method dengan prefix streamAll... dan return value Stream<T>

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    1 usage  new *
    Stream<Product> streamAllByCategory(Category category);
```

```java
    @Test
    void stream() {
        transactionOperations.executeWithoutResult(transactionStatus -> {
            Category category = categoryRepository.findById(1L).orElse(null);
            assertNotNull(category);

            Stream<Product> stream = productRepository.streamAllByCategory(category);
            stream.forEach(product -> System.out.println(product.getId() + " : " + product.getName()));
        });
```

Slice

Slice

- Saat kita mengembalikan data dalam bentuk Page<T>, maka kita hanya akan dapat data untuk nomor page yang dipilih
- Kita bisa menggunakan Slice<T>, yang bisa mengembalikan informasi apakah ada next page dan previous page
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Slice.html

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    no usages  new *
    Slice<Product> findAllByCategory(Category category, Pageable pageable);
```

Kode : Test Product Repository

```java
@Test
void slice() {
    Pageable firstPage = PageRequest.of(0, 1);

    Category category = categoryRepository.findById(1L).orElse(null);
    assertNotNull(category);

    Slice<Product> slice = productRepository.findAllByCategory(category, firstPage);
    // do with content
    while (slice.hasNext()) {
        slice = productRepository.findAllByCategory(category, slice.nextPageable());
        // do with content
    }
```

Locking

Locking

- Di kelas JPA, kita sudah bahas melakukan Pessimistic Locking
- Karena di Spring Data JPA, kita tidak perlu lagi menggunakan Entity Manager, bagaimana jika kita butuh melakukan Pessimistic Locking?
- Kita bisa membuat Query Method dengan menambahkan annotation @Lock
- https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/Lock.html

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> {

    no usages  new *
    @Lock(LockModeType.PESSIMISTIC_WRITE)
    Optional<Product> findFirstByIdEquals(Long id);
```

Kode : Test Product Repository

```java
@Test
void lock1() {
    transactionOperations.executeWithoutResult(transactionStatus -> {
        try {
            Product product = productRepository.findFirstByIdEquals(1L)
                    .orElse(null);
            assertNotNull(product);
            product.setPrice(30_000_000L);

            Thread.sleep(20_000L);
            productRepository.save(product);
        } catch (InterruptedException exception) {
            throw new RuntimeException(exception);
        }
    });
}
```

```java
@Test
void lock2() {
    transactionOperations.executeWithoutResult(transactionStatus -> {
        Product product = productRepository.findFirstByIdEquals(1L)
                .orElse(null);
        assertNotNull(product);
        product.setPrice(10_000_000L);
        productRepository.save(product);
    });
}
```

Auditing

Auditing

- Saat kita membuat table, sering sekali kita menambahkan informasi audit seperti createdAt dan updatedAt
- Spring Data JPA mendukung mengubahan data audit secara otomatis ketika proses save
- Kita cukup gunakan annotation @CreatedDate dan @LastModifiedDate, dan menggunakan EntityListener AuditingEntityListener
- Kita bisa menggunakan tipe data Date, Timestamp, Instance atau Long (milis) untuk field audit nya
- Secara default, fitur ini tidak aktif, untuk mengaktifkannya, kita harus menambahkan annotation @EnableJpaAuditing

Kode : Jpa Auditing

Kode : Alter Table Category

```sql
ALTER TABLE categories
    ADD COLUMN created_date TIMESTAMP;

ALTER TABLE categories
    ADD COLUMN last_modified_date TIMESTAMP;
```

Kode : Class Category

```java
@Table(name = "categories")
@EntityListeners({AuditingEntityListener.class})
public class Category {

    @CreatedDate
    @Column(name = "created_date")
    private Instant createdDate;

    @LastModifiedDate
    @Column(name = "last_modified_date")
    private Instant lastModifiedDate;
```

Kode : Test Category Repository

```java
@Test
void audit() {
    Category category = new Category();
    category.setName("Sample Audit");
    categoryRepository.save(category);

    assertNotNull(category.getId());
    assertNotNull(category.getCreatedDate());
    assertNotNull(category.getLastModifiedDate());
}
```

Example

Example

- Spring Data JPA memiliki fitur Query by Example, dimana kita bisa membuat data object Entity, lalu meminta Spring Data JPA untuk membuat Query berdasarkan data example Entity yang kita buat
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/Example.html

Example Repository

- JpaRepository memiliki parent interface QueryByExampleExecutor
- Dimana sudah disediakan banyak method yang bisa kita gunakan dengan parameter Example untuk mencari data
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/repository/query/QueryByExampleExecutor.html

Kode : Test Category Repository

```java
@Test
void example() {
    Category category = new Category();
    category.setName("GADGET MURAH");

    Example<Category> example = Example.of(category);

    List<Category> categories = categoryRepository.findAll(example);
    assertEquals(1, categories.size());
}
```

Example Matcher

- Example memiliki fitur Matcher, dimana kita bisa atur cara Example melakukan query
- https://docs.spring.io/spring-data/commons/docs/current/api/org/springframework/data/domain/ExampleMatcher.html

Kode : Test Category Repository

```java
@Test
void exampleMatcher() {
    Category category = new Category();
    category.setName("gadget murah");

    ExampleMatcher matcher = ExampleMatcher.matching().withIgnoreNullValues().withIgnoreCase();

    Example<Category> example = Example.of(category, matcher);

    List<Category> categories = categoryRepository.findAll(example);
    assertEquals(1, categories.size());
}
```

Specification

Specification Executor

- Di JPA, terdapat fitur Criteria untuk membuat Query secara dinamis
- Hal ini bisa kita gunakan fitur Specification di Spring Data JPA
- Untuk mendukung fitur ini, Repository yang kita buat harus extends JpaSpeficitaionExecutor, dimana terdapat banyak sekali method dengan parameter Specification
- https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/repository/JpaSpecificationExecutor.html

Specification

- Specification adalah lambda yang bisa kita buat dengan mengembalikan data JPA Predicate seperti yang perah kita pelajari di kelas JPA
- Kita bisa mendapatkan detail dari Root, CriteriaQuery dan CriteriaBuilder di method toPredicate() milik Specification
- https://docs.spring.io/spring-data/jpa/docs/current/api/org/springframework/data/jpa/domain/Specification.html

Kode : Product Repository

```
1 usage  new *
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> , JpaSpecificationExecutor<Product> {

}
```

Kode : Test Product Repository

```
@Test
void specification() {
    Specification<Product> specification = (root, criteria, builder) -> {
        return criteria.where(
                builder.or(
                        builder.equal(root.get("name"), "Apple iPhone 14 Pro Max"),
                        builder.equal(root.get("name"), "Apple iPhone 13 Pro Max")
                )
        ).getRestriction();
    };

    List<Product> products = productRepository.findAll(specification);
    assertEquals(2, products.size());
```

Projection

Projection

- Saat kita belajar JPA, kita tahu terdapat fitur di JPA QL untuk memanggil constructor sebuah class, sehingga return hasil query bisa dalam bentuk class bukan Entity
- Di Spring, terdapat fitur bernama Projection, yang mirip namun lebih mudah
- Caranya di Repository, kita bisa buat Query Method dengan return Interface yang kita inginkan, secara otomatis nanti Spring Data akan melakukan mapping sesuai dengan field hasil Query dengan Interface return nya
- Yup, tidak salah mengetik, jadi kita harus buat dalam bentuk Interface, bukan Class
- Hal ini agar Spring Data tahu bahwa itu adalah projection

Kode : Simple Product

```
4 usages  new
public interface SimpleProduct {

    no usages  new *
    Long getId();

    new *
    String getName();
}
    You, A minute ago • Uncommitted changes
```

Kode : Product Repository

```
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> , JpaSpecificationExecutor<Product> {

    no usages  new *
    List<SimpleProduct> findAllByNameLike(String name);
```

```
    @Test
    void projection() {
        List<SimpleProduct> simpleProducts = productRepository.findAllByNameLike("%Apple%");
        assertEquals(2, simpleProducts.size());
    }
```

Java Record

- Atau, jika sudah menggunakan versi Java 17, ada baiknya kita buat Projection dalam bentuk Java Record
- Bedanya dengan interface, saat menggunakan interface, maka Spring Data akan menggunakan Proxy (Reflection)
- Sedangkan ketika menggunakan Java Record, akan dibuat instance nya secara otomatis

Kode : Simple Product

```
4 usages  new *
public record SimpleProduct(Long id, String name) {
}
   You, Moments ago • Uncommitted changes
```

Dynamic Projection

- Kadang kita mungkin ingin membuat beberapa jenis Projection Interface / Record
- Pada kasus ini, kita bisa menggunakan Generic di Query Method nya, dan juga menambahkan parameter Class di parameter terakhir Query Method nya

Kode : Product Price

```
no usages  new *
public record ProductPrice(Long id, Long price) {
}
   You, Moments ago • Uncommitted changes
```

Kode : Product Repository

```java
@Repository
public interface ProductRepository extends JpaRepository<Product, Long> , JpaSpecificationExecutor<Product> {

    no usages  new *
    <T> List<T> findAllByNameLike(String name, Class<T> tClass);
```

```java
        List<SimpleProduct> simpleProducts = productRepository.findAllByNameLike("%Apple%", SimpleProduct.class);
        assertEquals(2, simpleProducts.size());

        List<ProductPrice> productPrices = productRepository.findAllByNameLike("%Apple%", ProductPrice.class);
        assertEquals(2, productPrices.size());
    }
```