Back End Development 1

Sesi 15

Introduction of Springboot part 1

Spring JPA

Spring data JPA merupakan salah satu yang mengimplementasikan JPA juga dan yang di custom sehingga operasi-operasi standar seperti Create Read Update Delete sudah disediakan.

Spring data JPA menggunakan interface Repository yang mana menggunakan ID dari class domain sebagai parameter nya. Disamping itu Spring data JPA mendukung query method, native query, JPQL dan lain sebagainya.

Membuat Projek Spring Data JPA Akses ke https://start.spring.io/ dan buat projek Spring Boot baru dengan spesifikasi di bawah ini :

Project	**Maven Project**
Versi Spring Boot	2.1.12
Group	id.belajar
Artifact	spring-data-jpa
Dependencies	Spring Data JPA
	MySQL Driver

Spring JPA

Jika kita lihat di file pom.xml, terlihat kita menggunakan dependencies

pom.xml

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

Dengan plugin ini, kita tidak perlu menambahkan lagi hibernate-core, spring-orm ke dalam projek secara manual. Semua nya sudah di bundle dalam plugin ini

pom.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <parent>
      <groupId>org.springframework.boot</groupId>
      <artifactId>spring-boot-starter-parent</artifactId>
      <version>2.2.4.RELEASE</version>
      <relativePath/> <!-- lookup parent from repository -->
   </parent>
   <groupId>id.belajar</groupId>
   <artifactId>spring-data-jpa</artifactId>
   <version>0.0.1-SNAPSHOT</version>
   <name>spring-data-jpa</name>
   <description>Demo project for Spring Boot</description>

   <properties>
      <java.version>1.8</java.version>
   </properties>

   <dependencies>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-data-jpa</artifactId>
      </dependency>
      <dependency>
         <groupId>com.zaxxer</groupId>
         <artifactId>HikariCP</artifactId>
         <version>3.1.0</version>
      </dependency>
      <dependency>
         <groupId>mysql</groupId>
         <artifactId>mysql-connector-java</artifactId>
         <scope>runtime</scope>
      </dependency>
      <dependency>
         <groupId>org.springframework.boot</groupId>
         <artifactId>spring-boot-starter-test</artifactId>
         <scope>test</scope>
         <exclusions>
            <exclusion>
               <groupId>org.junit.vintage</groupId>
               <artifactId>junit-vintage-engine</artifactId>
            </exclusion>
```

```
            </exclusions>
        </dependency>
    </dependencies>

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

</project>
```

Membuat Entity Class Spring JPA bekerja bekerja dengan class yang merepresentasikan entitas dalam sebuah system. Sebagai demo, buatkan sebuah class Entitas Book sebagai Berikut.

id.belajar.springdatajpa.entity.Book

```java
@Entity
public class Book {

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

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String writer;

    @Column(nullable = false)
    private String isbn;

    public Book() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }
```

```
    public String getWriter() {
        return writer;
    }

    public void setWriter(String writer) {
        this.writer = writer;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }
}
```

Penjelasan

Penjelasan dari class Book diatas :

@Entity : merepresentasikan bahwa class tersebut adalah sebuah entitas, yang nantinya akan ada sebuah Tabel di database dengan nama Book

@Id : menunjukan bahwa variable itu adalah sebuah ID atau Primary Key dari suatu table. Penggunaan @Id di Spring Data JPA sangat lah mandatory /wajib. Setiap class yang diberi annotasi @Entity harus mempunyai @Id.

@GeneratedValue : sebuah mekanisme dalam memberikan nilai. Karena disini menggunakan strategy = GenerationType.IDENTITY yang artinya nilai akan diberikan secara increment atau fungsi nya sama dengan AUTO_INCREMENT pada sebuah table.

@Column(nullable = false) : Memberikan info bahwa variable tersebut yang akan menjadi kolom di table Book, tidak boleh null atau not null.

Membuat JPA Properties dan Hikari Connection Pool

Sebenarnya Spring Boot versi 2 sudah secara default menggunakan HikariCP sebagai connection pool nya sebelumnya menggunakan tomcat pool. HIkariCP sangatlah cepat, ringan, dapat diandalkan dalam fase production.

Buka file application.properties di dalam path src/main/resources kemudian tambahkan beberapa konfigurasi dibawah ini dan pastikan Anda sudah membuat database dengan nama spring-data-jpa.

src.main.resources.application.properties

```
#MySQL Connection
spring.datasource.url = jdbc:mysql://localhost:3306/spring-data-jpa
spring.datasource.username = root
spring.datasource.password = root

#HikariCP
spring.datasource.hikari.connection-timeout=20000
spring.datasource.hikari.minimum-idle=5
spring.datasource.hikari.maximum-pool-size=12
spring.datasource.hikari.idle-timeout=300000
```

```
spring.datasource.hikari.max-lifetime=120000
spring.datasource.hikari.auto-commit=true
```

```
#JPA Properties
spring.jpa.database-platform=org.hibernate.dialect.MySQL5InnoDBDialect
spring.jpa.hibernate.ddl-auto=update
spring.datasource.driverClassName=com.mysql.jdbc.Driver
```

Penjelasan dari potongan kode diatas : spring.datasource.url : adalah alamat dari database MySQL yang sudah diinstall. Default nya yaitu localhost:3306

spring.datasource.username : adalah username untuk dapat mengakses database yang dalam hal ini kita menggunakan username sebagai root.

spring.datasource.password : adalah password yang digunakan untuk mengakses database dengan username diatas.

Tidak disarankan untuk menggunakan username root dan password root jika aplikasi kalian sudah release

spring.datasource.hikari.connection-timeout=20000 : adalah nilai maksimum dalam hitungan mili detik yang dibutuhkan client menunggu untuk sebuah connection dari connection pool.

spring.datasource.hikari.minimum-idle=5 : nilai atau jumlah dari koneksi idle

spring.datasource.hikari.maximum-pool-size=12 : adalah jumlah maksimum dari ukuran connection pool

spring.datasource.hikari.idle-timeout=300000 : jumlah maksimum dalam hitungan mili detik yang mengijinkan koneksi berada dalam status idle dalam connection pool

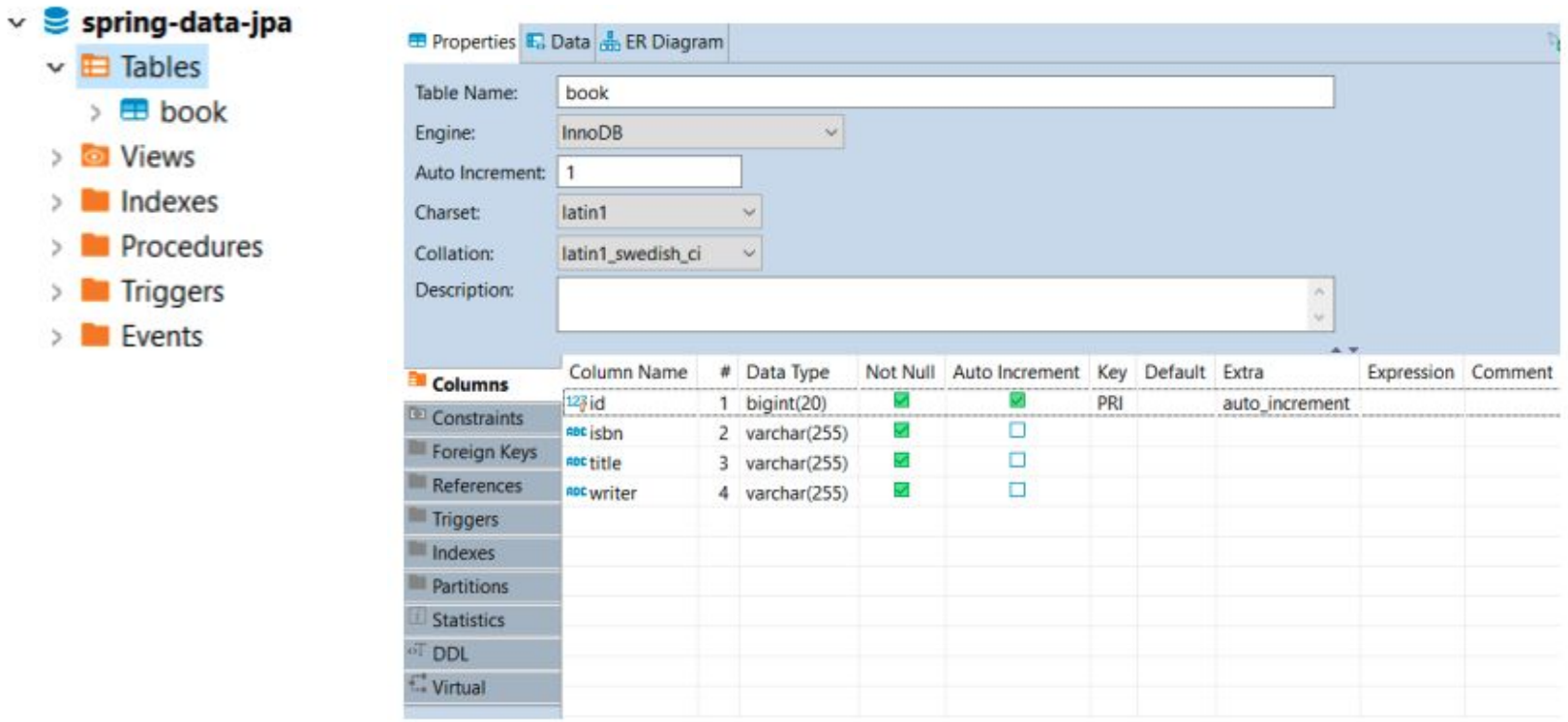
spring.datasource.hikari.max-lifetime=120000 : adalah maksimum waktu hidup (life time) dalam hitungan mili detik dari sebuah koneksi dalam connection pool setelah keluar.

spring.datasource.hikari.auto-commit=true : mengatur setting auto-commit dengan nilai nya true

spring.jpa.database-platform : dialek yang digunakan nantinya oleh JPA. Disini kita mengunakan dialek MySQL dengan tipe InnoDB.

spring.jpa.hibernate.ddl-auto : property ini bertugas untuk membuatkan table-tabel didatabase tergantung dari value nya. Disini kita menggunakan value update yang artinya Spring Data JPA akan secara otomatis membuatkan table Book di database.

spring.datasource.driverClassName : sesuai dengan nama nya, property ini digunakan untuk menentukan driver yang digunakan sebagai jembatan komunikasi dengan database. Jalankan program ini jika tidak terdapat error, kita bisa lihat di DBeaver pada database spring-data-jpa akan terdapat Book table.



HACKTIV8

Kita bisa lihat apa yang kita sudah deklarasikan di class Book maka akan di representasikan menjadi sebuah table. Untuk kolom Id menjadi primary key dengan nilai auto_increment. Dan untuk kolom isbn, title, writer mempunyai atribut Not Null.

Eksekusi Query Method Untuk dapat menggunakan fitur query method bawaan Spring Data JPA maka kita harus membuat Layer Repository untuk class Book.

id.belajar.springdatajpa.repository.BookRepository

```
import id.belajar.springdatajpa.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}
```

Dari kode diatas dapat kita lihat bahwa BookRepository adalah turunan JPARepository dimana JPARepository menyediakan fungsi untuk create, retrieve, update, delete, findAll dan lain-lain sehingga kita tidak perlu lagi untuk membuat fungsi-fungsi CRUD dari awal.

Supaya lebih paham kita akan coba menyimpan dua buah object Book ke dalam database.

id.belajar.springdatajpa.SpringDataJpaApplication

```java
@SpringBootApplication
public class SpringDataJpaApplication implements CommandLineRunner {

   private final Logger LOG =
LoggerFactory.getLogger(SpringDataJpaApplication.class);

   @Autowired
   private BookRepository bookRepository;

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

   @Override
   public void run(String... args) throws Exception {

      Book book1 = new Book();
      book1.setTitle("Belajar Spring Boot");
      book1.setWriter("Teten Nugraha");
      book1.setIsbn("IS-90908");

      Book book2 = new Book();
      book2.setTitle("Belajar Spring Boot 2");
      book2.setWriter("Teten Nugraha");
      book2.setIsbn("IS-9090890");

      bookRepository.save(book1);
      bookRepository.save(book2);

      LOG.info("Berhasil menyimpan "+book1);
      LOG.info("Berhasil menyimpan "+book2);

   }
}
```

Jika kita jalankan kode diatas, maka dalam console LOG aka nada keterangan mengenai dua object Book yang sudah kita simpan ke database.

```
INFO 8976 --- [           main] i.b.s.SpringDataJpaApplication           : Berhasil menyimpan Book{id=1, title='Belajar Spring Boot', writer='Teten Nugraha', isbn='IS-90908'}
INFO 8976 --- [           main] i.b.s.SpringDataJpaApplication           : Berhasil menyimpan Book{id=2, title='Belajar Spring Boot 2', writer='Teten Nugraha', isbn='IS-9090890'}
```

Disamping method save kita akan coba untuk meretrieve data. Edit kembali file BookRepository dan tambahkan query methodnya sebagai berikut.

id.belajar.springdatajpa.repository.BookRepository

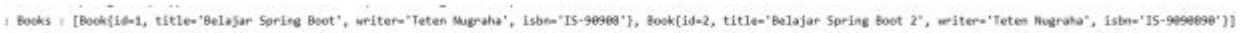
```
List<Book> findAll();
```

Kita ubah kembali class SpringDataJpaApplication dan panggil method findAll diatas.

id.belajar.springdatajpa.SpringDataJpaApplication

```
@SpringBootApplication
public class SpringDataJpaApplication implements CommandLineRunner {

   private final Logger LOG =
LoggerFactory.getLogger(SpringDataJpaApplication.class);

   @Autowired
   private BookRepository bookRepository;

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

   @Override
   public void run(String... args) throws Exception {

      List<Book> books = bookRepository.findAll();

      LOG.info("Books : "+books);

   }
}
```

Kita jalankan kode nya, terlihat di consle log terdapat dua buah nilai dari hasil method findAll.



Disamping findAll, kita juga bisa membuat query misalnya untuk mendapatkan data book berdasarkan penulis atau writer.

id.belajar.springdatajpa.repository.BookRepository

```
List<Book> findAllByWriter(String writer);
```

id.belajar.springdatajpa.SpringDataJpaApplication

```
@Override
public void run(String... args) throws Exception {

    final String writer = "Teten Nugraha";

    List<Book> books = bookRepository.findAllByWriter(writer);

    LOG.info("Books : "+books);

}
```

Kita jalankan kembali dan lihat log nya.

```
: Books : [Book{id=1, title='Belajar Spring Boot', writer='Teten Nugraha', isbn='IS-90908'}, Book{id=2, title='Belajar Spring Boot 2', writer='Teten Nugraha', isbn='IS-9090890'}]
```

Terakhir, misalkan kita ingin melakukan pencarian data Book berdasarkan nomor ISBN. Nomor ISBN itu unik dan pasti akan ada satu data Book dengan satu nomor ISBN, oleh karenanya kita buat method seperti ini.

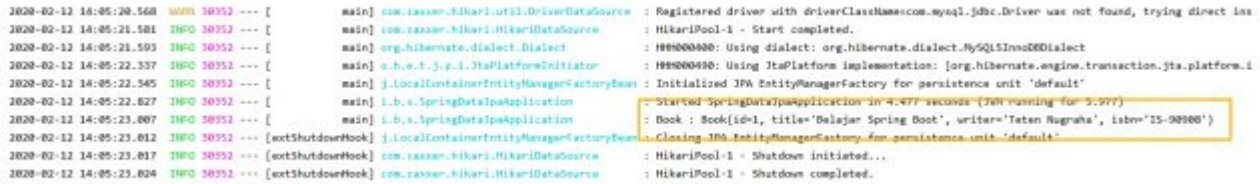
id.belajar.springdatajpa.repository.BookRepository

```
Book findByIsbn(String isbn);
```

id.belajar.springdatajpa.SpringDataJpaApplication

```
@Override
public void run(String... args) throws Exception {

    final String isbn = "IS-90908";

    Book book = bookRepository.findByIsbn(isbn);

    LOG.info("Book : "+book);

}
```

Jalankan, kita cek kembali di log nya



Berikut kode lengkap dari BookRepository.

id.belajar.springdatajpa.repository.BookRepository

```
public interface BookRepository extends JpaRepository<Book, Long> {

    List<Book> findAll();

    List<Book> findAllByWriter(String writer);

    Book findByIsbn(String isbn);

    Book findByTitle(String title);
}
```

HACKTIV8

Eksekusi Native Query

Untuk beberapa kasus, kadang kita ingin menggunakan native sql syntax secara langsung seperti yang kita pernah gunakan jika menggunakan Statement atau PreparedStatement.

Spring Data juga sudah menyediakan untuk fungsi ini. Masih di file yang sama yaitu di file BookRepository.

id.belajar.springdatajpa.repository.BookRepository

```
public interface BookRepository extends JpaRepository<Book, Long> {

    ...

    @Query(
        nativeQuery = true,
        value = "select * from book"
    )
    List<Book> findAllQueryNative();

    ...
}
```

id.belajar.springdatajpa.SpringDataJpaApplication

```java
@Override
public void run(String... args) throws Exception {

    List<Book> books = bookRepository.findAllQueryNative();

    LOG.info("Book : "+books);

}
```

Jika kita running maka dalam log akan ada dua data book yang kita panggil.

```
DriverDataSource  : Registered driver with driverClassName=com.mysql.jdbc.Driver was not found, trying direct instantiation.
iDataSource       : HikariPool-1 - Start completed.
ialect            : HHH000400: Using dialect: org.hibernate.dialect.MySQL5InnoDBDialect
rmInitiator       : HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
anagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
ication           : Started SpringDataJpaApplication in 3.974 seconds (JVM running for 5.884)
ication           : Book : [Book{id=1, title='Belajar Spring Boot', writer='Teten Nugraha', isbn='IS-90908'}, Book{id=2, title='Belajar Spring Boot 2', writer='Teten Nugraha', isbn='IS-9090890'}]
anagerFactoryBean : Closing JPA EntityManagerFactory for persistence unit 'default'
iDataSource       : HikariPool-1 - Shutdown initiated...
iDataSource       : HikariPool-1 - Shutdown completed.
```

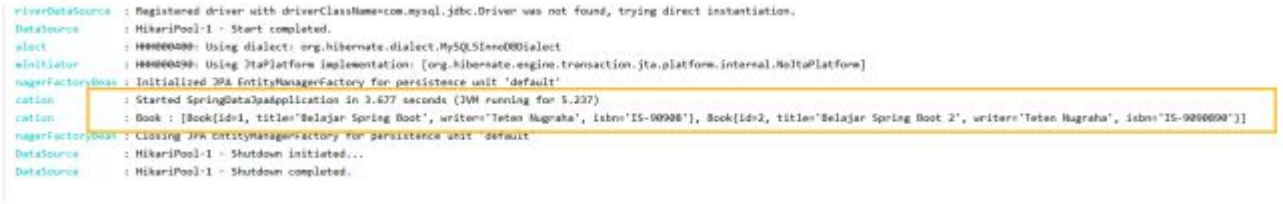
nah sekarang bagaimana cara nya jika kita ingin melempar sebuah parameter, katakanlah kita ingin mendapatkan semua data buku berdasarkan nama penulis ?.

id.belajar.springdatajpa.repository.BookRepository

```
public interface BookRepository extends JpaRepository<Book, Long> {

    ...

    @Query(
        nativeQuery = true,
        value = "select * from book where writer = ?1"
    )
    List<Book> findAllByWriterQueryNative(String writer);


    ...
}
```

id.belajar.springdatajpa.SpringDataJpaApplication

```
@Override
public void run(String... args) throws Exception {

   final String writer = "Teten Nugraha";

   List<Book> books = bookRepository.findAllByWriterQueryNative(writer);

   LOG.info("Book : "+books);

}
```



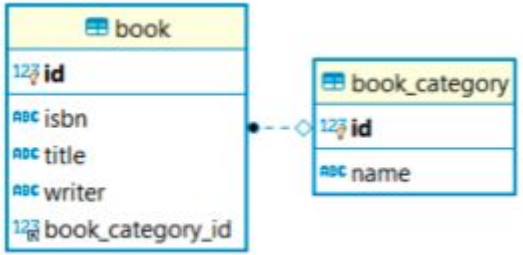
Untuk file BookRepository jika digabungkan akan menjadi seperti ini

id.belajar.springdatajpa.repository.BookRepository

```java
package id.belajar.springdatajpa.repository;

import id.belajar.springdatajpa.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface BookRepository extends JpaRepository<Book, Long> {

    List<Book> findAll();

    List<Book> findAllByWriter(String writer);

    Book findByIsbn(String isbn);

    Book findByTitle(String title);

    @Query(
            nativeQuery = true,
            value = "select * from book"
    )
    List<Book> findAllQueryNative();

    @Query(
            nativeQuery = true,
            value = "select * from book where writer = ?1"
    )
    List<Book> findAllByWriterQueryNative(String writer);
}
```

One to Many Relationship

One to Many relationsip adalah salah satu relasi yang menghubungkan antara dua entitas atau dua table yang pada umumnya sering disebut hubungan parent-child dimana atribut parent selalu ada pada child. Pada sesi ini, kita akan membuat perubahan enhancement pada class Book dan membuat satu class baru dengan nama BookCategory.



Buat Sebuah class BookCategory pada package entity. Class ini nantinya akan menjadi class parent untuk class Book beserta class BookCategoryRepositorynya.

id.belajar.springdatajpa.entity.BookCategory

```
package id.belajar.springdatajpa.entity;

import javax.persistence.*;
import java.util.List;
import java.util.stream.Collectors;
import java.util.stream.Stream;
```

```java
@Entity
public class BookCategory {

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

    private String name;

    @OneToMany(
            mappedBy = "bookCategory",
            cascade = CascadeType.ALL
    )
    private List<Book> books;

    public BookCategory() {
    }

    public BookCategory(String name, Book... books) {
        this.name = name;
        this.books = Stream.of(books).collect(Collectors.toList());
        this.books.forEach(x -> x.setBookCategory(this));
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

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

    public List<Book> getBooks() {
        return books;
    }

    public void setBooks(List<Book> books) {
        this.books = books;
    }

    @Override
    public String toString() {
        return "BookCategory{" +
```

```
                "id=" + id +
                ", name='" + name + '\'' +
                ", books=" + books +
                '}';
    }
}
```

Penjelasan kode diatas.
 - @OneToMany adalah annotasi yang yang mendefinisikan relasi one-to-many sehingga class BookCategory akan mempunyai banyak child Book.
- mappedBy adalah property yang memberitahu Spring Data JPA bahwa foreign key dari relasi ini berada pada class child pada variable bookCategory.
- CascadeType.ALL adalah sebuah tipe cascade yang akan menyebarkan semua operasi EntityManager (PERSIST, REMOVE, REFRESH, MERGE, DETACH) ke entitas yang berelasi. Selanjutnya, edit class Book dan tambahkan objek variable BookCategory bookCategory

```java
@ManyToOne
@JoinColumn
private BookCategory bookCategory;
```

- @ManyToOne mendefinisikan relasi many-to-one antara dua entitas.

- @JoinColumn mengindikasikan bahwa entitas Book adalah yang mempunyai kunci (foreign key) yang menghubungkan dengan class parent yaitu BookCategory.

id.belajar.springdatajpa.entity.Book

```java
package id.belajar.springdatajpa.entity;

import javax.persistence.*;
```

```java
@Entity
public class Book {

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

    @Column(nullable = false)
    private String title;

    @Column(nullable = false)
    private String writer;

    @Column(nullable = false)
    private String isbn;

    @ManyToOne
    @JoinColumn
```

```
    private BookCategory bookCategory;

    public Book() {
    }

    public Book(String title, String writer, String isbn) {
        this.title  = title;
        this.writer = writer;
        this.isbn = isbn;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getTitle() {
        return title;
    }

    public void setTitle(String title) {
        this.title = title;
    }

    public String getWriter() {
        return writer;
    }

    public void setWriter(String writer) {
        this.writer = writer;
    }

    public String getIsbn() {
        return isbn;
    }

    public void setIsbn(String isbn) {
        this.isbn = isbn;
    }

    public BookCategory getBookCategory() {
        return bookCategory;
    }

    public void setBookCategory(BookCategory bookCategory) {
        this.bookCategory = bookCategory;
    }
```

```
    @Override
    public String toString() {
        return "Book{" +
                "id=" + id +
                ", title='" + title + '\'' +
                ", writer='" + writer + '\'' +
                ", isbn='" + isbn + '\'' +
                '}';
    }
}
```

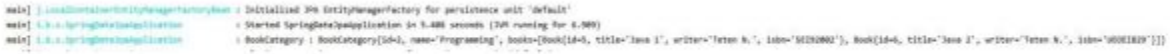
id.belajar.springdatajpa.repository.BookCategoryRepository

```
package id.belajar.springdatajpa.repository;

import id.belajar.springdatajpa.entity.BookCategory;
import org.springframework.data.jpa.repository.JpaRepository;

public interface BookCategoryRepository extends JpaRepository<BookCategory,
Long> {
}
```

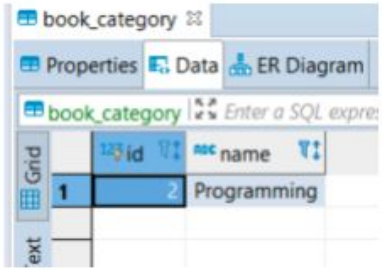
Untuk melakukan pengetesan, lakukan pada class SpringDataJpaApplication dan lakukan injeksi terhadap class BookCategoryRepository dan class BookRepository. Setelah itu kita masukan data BookCategory beserta dengan Book nya namun sebelumnya hapus terlebih dahulu data Book yang sudah ada didalam database.

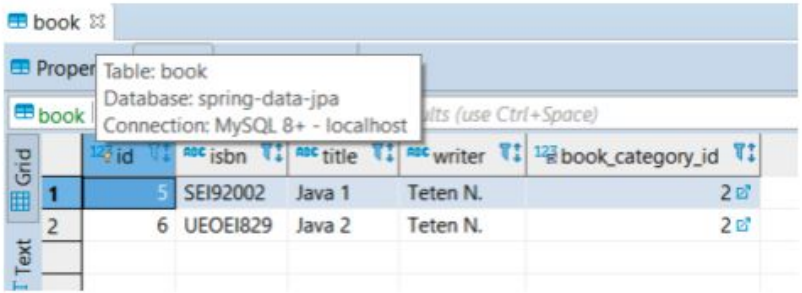
```
@Autowired
private BookRepository bookRepository;

@Autowired
private BookCategoryRepository bookCategoryRepository;

@Override
public void run(String... args) throws Exception {

   // create Book Category
   BookCategory bookCategory = bookCategoryRepository.save(new
BookCategory("Programming", new Book("Java 1","Teten N.","SEI92002"), new
Book("Java 2","Teten N.","UEOEI829")));

   LOG.info("BookCategory : "+bookCategory);

}
```

Jika dijalankan,maka dalam LOG akan terdapat keterangan data dari BookCategory beserta Book nya.



Jika kita check dalam database, hasilnya akan sama dengan yang di LOG





HACKTIV8

id.belajar.springdatajpa.SpringDataJpaApplication

```
@SpringBootApplication
public class SpringDataJpaApplication implements CommandLineRunner {

   private final Logger LOG =
LoggerFactory.getLogger(SpringDataJpaApplication.class);

   @Autowired
   private BookRepository bookRepository;

   @Autowired
   private BookCategoryRepository bookCategoryRepository;

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

   @Override
   public void run(String... args) throws Exception {

      // create Book Category
      BookCategory bookCategory = bookCategoryRepository.save(new
BookCategory("Programming", new Book("Java 1","Teten N.","SEI92002"), new
Book("Java 2","Teten N.","UEOEI829")));

      LOG.info("BookCategory : "+bookCategory);

   }
}
```

Many to Many Relationships

Setelah relasi one-to-many selanjutnya akan kita bahas mengenai relasi many-to-many dengan menggunakan contoh kasus relasi antara entitas students dan courses.

Dimana satu Students bisa memiliki bayak Courses dan begitu juga satu Courses dapat memiliki banyak Students. Biasanya relasi ini akan membuat sebuah table yang menghubungkan kedua entitas seperti pada diagram ERD berikut.

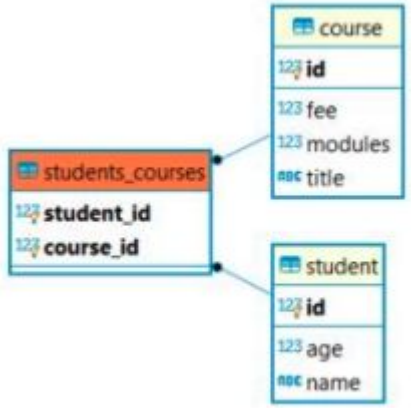


Table Students_courses diatas adalah sebuah table yang mempunyai dua foreign key, student_id dan course_id dimana student_id menghubungkan dengan table Student dan course_id menghubungkan dengan table Course.

Buat dua buah class dengan nama Student dan Course .

id.belajar.springdatajpa.entity.Student

```
@Entity
public class Student {

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

    private String name;
    private int age;

    @ManyToMany(
            fetch = FetchType.LAZY,
            cascade = CascadeType.PERSIST
    )
    @JoinTable(
            name = "students_courses",
            joinColumns = {
                    @JoinColumn(name = "student_id", referencedColumnName =
"id", nullable = false, updatable = false)},
            inverseJoinColumns = {
                    @JoinColumn(name = "course_id", referencedColumnName = "id",
nullable = false, updatable = false)
            }
    )
    private Set<Course> courses = new HashSet<>();
```

```
    public Student() {
    }

    public Student(String name, int age) {
      this.name = name;
      this.age = age;
    }


    // Setter, Getter and toString
}
```

Penjelasan kode diatas.
- FetchType.LAZY konfigurasi yang tidak secara otomatis memuat data yang berada pada relasi nya sehingga untuk memuat harus menggunakan getter method. |
- CascadeType.PERSIST artinya bahwa operasi save() dari Spring Data JPA cascade ke entitas yang berelasi.
- @JoinTable adalah anotasi yang akan membuat sebuah table baru dengan nama students_courses sesuai pada gambar ERD diatas. Menggunakan @JoinColumn untuk menghubungkan antara foreign_key dengan table referensi nya.

id.belajar.springdatajpa.entity.Course

```java
@Entity
public class Course {

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

    private String title;
    private int modules;
    private double fee;

    @ManyToMany(
            mappedBy = "courses",
            fetch = FetchType.LAZY
    )
    private Set<Student> students = new HashSet<>();

    public Course() {
    }

    public Course(String title, int modules, double fee) {
      this.title = title;
      this.modules = modules;
      this.fee = fee;
    }


    // Setter, Getter and toString()
}
```

id.belajar.springdatajpa.repository.StudentRepository

```java
package id.belajar.springdatajpa.repository;

import id.belajar.springdatajpa.entity.Student;
import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<Student, Long> {
}
```

id.belajar.springdatajpa.repository.CourseRepository

```java
package id.belajar.springdatajpa.repository;

import id.belajar.springdatajpa.entity.Course;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CourseRepository extends JpaRepository<Course, Long> {
}
```

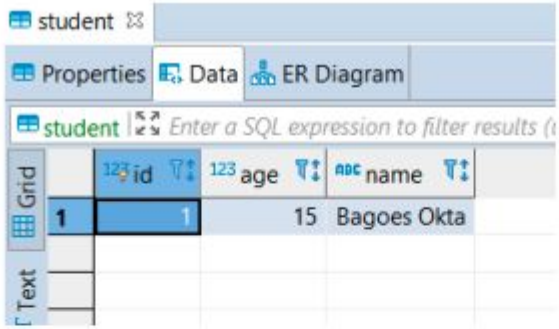
Pada class SpringDataJPAApplication lakukan injeksi pada class StudentRepository dan CourseRepository lalu buat object student dan course seperti pada source code berikut.

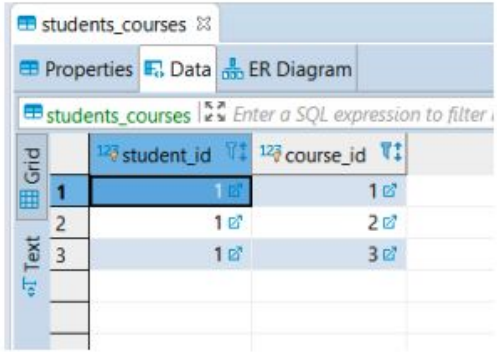
id.belajar.springdatajpa.SpringDataJpaApplication

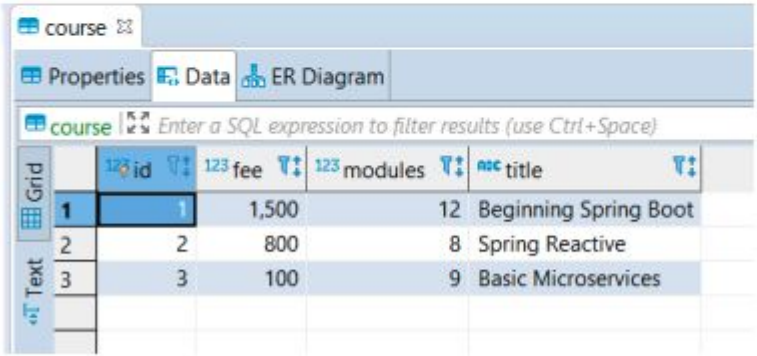
```java
@SpringBootApplication
public class SpringDataJpaApplication implements CommandLineRunner {

   private final Logger LOG =
LoggerFactory.getLogger(SpringDataJpaApplication.class);

   @Autowired
   private StudentRepository studentRepository;

   @Autowired
   private CourseRepository courseRepository;

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

   @Override
   public void run(String... args) throws Exception {

      // create a student
      Student student = new Student("Bagoes Okta", 15);

      // save the student
      studentRepository.save(student);
```

```
        // create three courses
        Course course1 = new Course("Beginning Spring Boot",  12, 1500);
        Course course2 = new Course("Spring Reactive", 8, 800);
        Course course3 = new Course("Basic Microservices",9, 100);

        // save courses
        courseRepository.saveAll(Arrays.asList(course1, course2, course3));

        // add courses to the student
        student.getCourses().addAll(Arrays.asList(course1, course2, course3));

        // update the student
        studentRepository.save(student);

    }
}
```

Jalankan programnya, lalu cek menggunakan DBeaver pada table students, courses, dan students_courses.







HACKTIV8