Final Project

Title: Book Store

Summary:

This project consists of a book store and its common entities. The entities are book, publisher and genre.

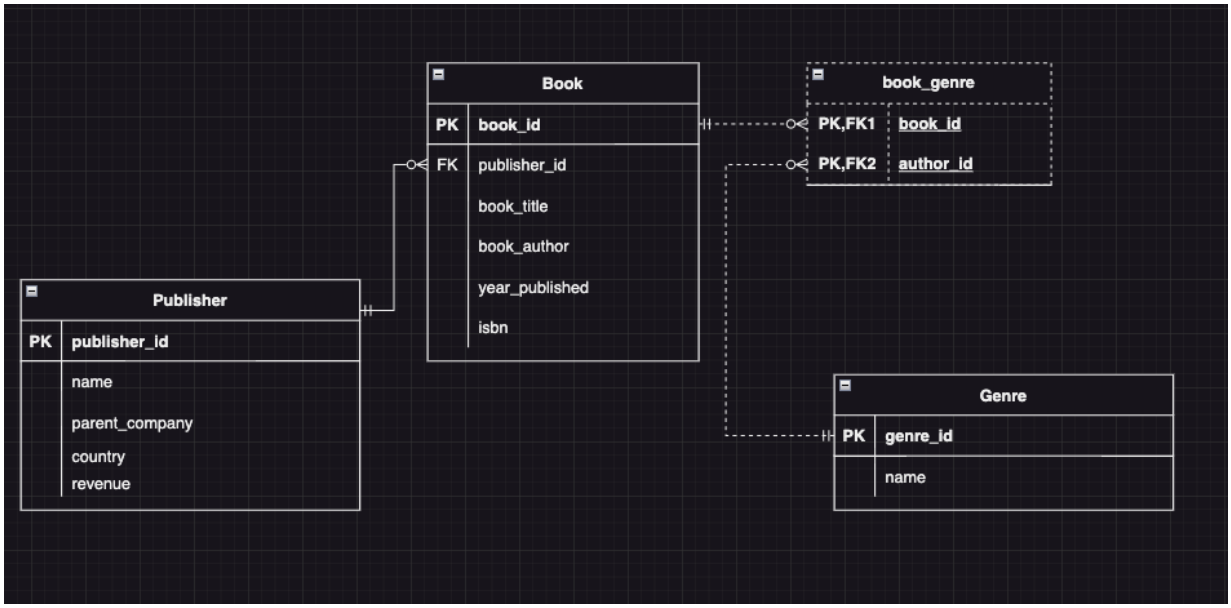
Relationships:

- Publisher has a one to many relationship with book
- Book has a many to many relationship with genre.

CRUD OPERATIONS:

- Publisher:
  - Create through the "/publisher" URL
  - Read through the "/publisher" and "/publisher/{publisherId}" URLs
  - Update through the "/publisher/{publisherId}" URL
  - Delete through the "/publisher/{publisherId}" URL
- Book:
  - Create through the "/publisher/{publisherId}/book" URL
  - Read through the "/publisher/{publisherId}/book/{bookId}" URL
  - Update through the "/publisher/{publisherId}/book/{bookId}" URL
  - Delete through the "/publisher/{publisherId}/book/{bookId} URL
- Genre:
  - Update through the "/genre" URL
  - Delete through the "/genre/{genreId}" URL

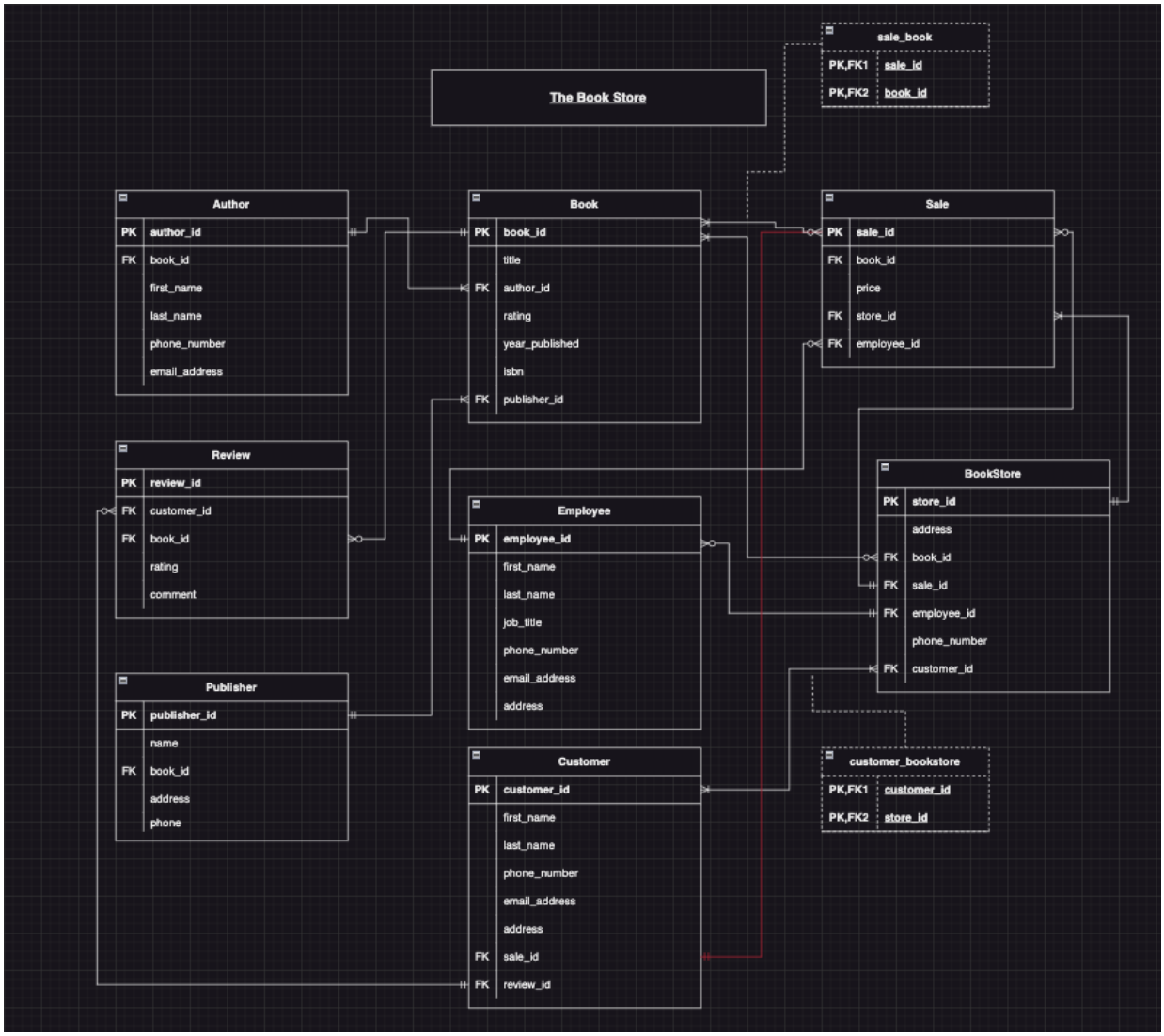
ERD:



Stretch Goals (to be completed if time allows, or after graduation):

I would like add more entities such as stores, employees, sales, reviews, ratings and authors. I would also like to build the front end and make a full application to deliver to one of my local used book stores in town.

Future ERD:



CODE:

BookStoreApplication

```
package bookstore;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BookStoreApplication {
public static void main(String[] args) {
SpringApplication.run(BookStoreApplication.class, args);

}

}
```

BookStoreController

```
package bookstore.controller;

import java.util.List;
import java.util.Map;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;

import bookstore.controller.model.BookData;
import bookstore.controller.model.PublisherData;
import bookstore.entity.Genre;
import bookstore.service.BookStoreService;
import lombok.extern.slf4j.Slf4j;

@RestController
@RequestMapping("/book_store")
@Slf4j
public class BookStoreController {
@Autowired
private BookStoreService bookStoreService;

//PUBLISHER ROUTES
@PostMapping("/publisher")
@ResponseStatus(code = HttpStatus.CREATED)
public PublisherData createPublisher(@RequestBody PublisherData publisherData) {
log.info("Creating a publisher {}", publisherData);
return bookStoreService.savePublisher(publisherData);
}
@PutMapping("/publisher/{publisherId}")
public PublisherData updatePublisher(@PathVariable Long publisherId,
@RequestBody PublisherData publisherData) {
publisherData.setPublisherId(publisherId);
log.info("Updating publisher{}", publisherData);
return bookStoreService.savePublisher(publisherData);
}
```

```
public List<PublisherData> retrieveAllPublishers() {
log.info("Retrieve all publishers called.");
return bookStoreService.retrieveAllPublishers();
}

@GetMapping("/publisher/{publisherId}")
public PublisherData retrievePublisherById(@PathVariable Long publisherId) {
log.info("Retrieving publisher with ID={}", publisherId);
return bookStoreService.retrievePublisherById(publisherId);
}

@DeleteMapping("/publisher")
public void deleteAllPublishers() {
log.info("Attempting to delete all publishers.");
throw new UnsupportedOperationException("Deleting all publishers is NOT allowed.");
}

@DeleteMapping("/publisher/{publisherId}")
public Map<String, String> deletePublisherById(@PathVariable Long publisherId) {
log.info("Deleting publisher with ID= {}" + publisherId);

bookStoreService.deletePublisherById(publisherId);

return Map.of("message", "Deletion the publisher with ID= " + publisherId + " was
successfull.");

}
//BOOK ROUTES

@PostMapping("/publisher/{publisherId}/book")
@ResponseStatus(code = HttpStatus.CREATED)
public BookData insertBook(@PathVariable Long publisherId, @RequestBody BookData
bookData) {
log.info("Creating book {} for publisher with ID= {}", bookData, publisherId);
return bookStoreService.saveBook(publisherId, bookData);
}
@GetMapping("/publisher/{publisherId}/book/{bookId}")
public BookData retrieveBookById(@PathVariable Long publisherId,
@PathVariable Long bookId){
log.info("Retrieving book with ID= {} for publisher with ID= {}", bookId,
publisherId);
return bookStoreService.retrieveBookById(publisherId, bookId);
}
@PutMapping("/publisher/{publisherId}/book/{bookId}")
public BookData updateBook(@PathVariable Long publisherId,
@PathVariable Long bookId,
@RequestBody BookData bookData) {
```

```
bookData.setBookId(bookId);
log.info("Creating book {} for publisher with ID= {}", bookData,publisherId);

return bookStoreService.saveBook(publisherId, bookData);
}
@DeleteMapping("/book/{bookId}")
public Map<String, String> deleteBookById(@PathVariable Long bookId) {
log.info("Deleting book with ID= {}" + bookId);

bookStoreService.deleteBookById(bookId);

return Map.of("message", "Deletion the book with ID= " + bookId + " was
successfull.");

}
@DeleteMapping("/book")
public void deleteAllBooks() {
log.info("Attempting to delete all books.");
throw new UnsupportedOperationException("Deleting all books is NOT allowed.");
}
//GENRE ROUTES
@GetMapping("/genre")
public List<Genre> retrieveAllGenre() {
log.info("Retrieve all genre called.");
return bookStoreService.retrieveAllGenres();
}
@DeleteMapping("/genre/{genreId}")
public Map<String, String> deleteGenreById(@PathVariable Long genreId) {
log.info("Deleting genre with ID= {}" + genreId);

bookStoreService.deleteGenreById(genreId);

return Map.of("message", "Deletion the genre with ID= " + genreId + " was
successfull.");

}
@DeleteMapping("/genre")
public void deleteAllGenres() {
log.info("Attempting to delete all genres.");
throw new UnsupportedOperationException("Deleting all genres is NOT allowed.");
}
}
```

GlobalControllerErrorHandler

```
import java.time.ZonedDateTime;
import java.time.format.DateTimeFormatter;
import java.util.NoSuchElementException;

import org.springframework.dao.DuplicateKeyException;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestControllerAdvice;
import org.springframework.web.context.request.ServletWebRequest;
import org.springframework.web.context.request.WebRequest;

import lombok.Data;
import lombok.extern.slf4j.Slf4j;

@RestControllerAdvice
@Slf4j
public class GlobalControllerErrorHandler {

private enum LogStatus {
STACK_TRACE, MESSAGE_ONLY
}

// creating getters and setters
@Data
private class ExceptionMessage {
private String message;
private String statusReason;
private int statusCode;
private String timeStamp;
private String uri;
}
@ExceptionHandler(IllegalStateException.class)
@ResponseStatus(code = HttpStatus.BAD_REQUEST)
public ExceptionMessage handleIllegalStateException(IllegalStateException exception,
WebRequest webRequest) {
return buildExceptionMessage(exception, HttpStatus.BAD_REQUEST, webRequest,
LogStatus.MESSAGE_ONLY);
}
@ExceptionHandler(UnsupportedOperationException.class)
@ResponseStatus(code = HttpStatus.METHOD_NOT_ALLOWED)
public ExceptionMessage
hadleUnsupportedOPerationException(UnsupportedOperationException exception,
WebRequest webRequest) {
return buildExceptionMessage(exception, HttpStatus.METHOD_NOT_ALLOWED, webRequest,
LogStatus.MESSAGE_ONLY);
```

```
}

@ExceptionHandler(Exception.class)
@ResponseStatus(code = HttpStatus.INTERNAL_SERVER_ERROR)
public ExceptionMessage handleException(Exception exception, WebRequest webRequest)
{
return buildExceptionMessage(exception, HttpStatus.INTERNAL_SERVER_ERROR,
webRequest, LogStatus.STACK_TRACE);

}

@ExceptionHandler(NoSuchElementException.class)
@ResponseStatus(code = HttpStatus.NOT_FOUND)
public ExceptionMessage handleNoSuchElementException(NoSuchElementException
exception, WebRequest webRequest) {
return buildExceptionMessage(exception, HttpStatus.NOT_FOUND, webRequest,
LogStatus.MESSAGE_ONLY);
}

@ExceptionHandler(DuplicateKeyException.class)
@ResponseStatus(code = HttpStatus.CONFLICT)
public ExceptionMessage handleDuplicateKeyException(DuplicateKeyException exception,
WebRequest webRequest) {
return buildExceptionMessage(exception, HttpStatus.CONFLICT, webRequest,
LogStatus.MESSAGE_ONLY);
}

private ExceptionMessage buildExceptionMessage(Exception exception, HttpStatus
status, WebRequest webRequest,
LogStatus logStatus) {

String message = exception.toString();
String statusReason = status.getReasonPhrase();
int statusCode = status.value();
String timeStamp = ZonedDateTime.now().format(DateTimeFormatter.RFC_1123_DATE_TIME);
String uri = null;

if (webRequest instanceof ServletWebRequest swr) {
uri = swr.getRequest().getRequestURI();
}

if (logStatus == LogStatus.MESSAGE_ONLY) {
log.error("Exception: {})", exception.toString());
} else {
log.error("Exception: ", exception);
}
```

```
ExceptionMessage exceptionMessage = new ExceptionMessage();

exceptionMessage.setMessage(message);
exceptionMessage.setStatusCode(statusCode);
exceptionMessage.setStatusReason(statusReason);
exceptionMessage.setTimeStamp(timeStamp);
exceptionMessage.setUri(uri);

return exceptionMessage;
}
}
```

BookData

```
package bookstore.controller.model;

import java.util.HashSet;
import java.util.Set;

import bookstore.entity.Book;
import bookstore.entity.Genre;
import bookstore.entity.Publisher;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
public class BookData {
private Long bookId;
private String bookTitle;
private String bookAuthor;
private String yearPublished;
private String isbn;
private BookPublisher publisher;
private Set<String> genres = new HashSet<String>();

public BookData(Book book) {
bookId = book.getBookId();
bookTitle = book.getBookTitle();
bookAuthor = book.getBookAuthor();
yearPublished = book.getYearPublished();
isbn = book.getIsbn();
publisher = new BookPublisher(book.getPublisher());

for (Genre genre : book.getGenres()) {
genres.add(genre.getName());
```

```
}
}

@Data
@NoArgsConstructor
public static class BookPublisher {
private Long publisherId;
private String name;
private String parentCompany;
private String country;
private String revenue;

public BookPublisher(Publisher publisher) {
publisherId = publisher.getPublisherId();
name = publisher.getName();
parentCompany = publisher.getParentCompany();
country = publisher.getCountry();
revenue = publisher.getRevenue();
}
}
}
```

GenreData

```
package bookstore.controller.model;

import java.util.HashSet;
import java.util.Set;

import bookstore.entity.Genre;
import bookstore.entity.Book;
import lombok.Data;
import lombok.NoArgsConstructor;
@Data
@NoArgsConstructor
public class GenreData {
private Long genreId;
private String name;
public Set<String> books = new HashSet<String>();
public GenreData(Genre genre) {
genreId = genre.getGenreId();
this.name = genre.getName();
for(Book book: genre.getBooks()) {
books.add(book.getBookTitle());
}
}
```

```
}
```

PublisherData

```
package bookstore.controller.model;

import java.util.HashSet;
import java.util.Set;

import bookstore.entity.Book;
import bookstore.entity.Genre;
import bookstore.entity.Publisher;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
public class PublisherData {
private Long publisherId;
private String name;
private String parentCompany;
private String country;
private String revenue;
private Set<BookResponse> books = new HashSet<BookResponse>();
public PublisherData(Publisher publisher) {
this.publisherId = publisher.getPublisherId();
this.name = publisher.getName();
this.parentCompany = publisher.getParentCompany();
this.country = publisher.getCountry();
this.revenue = publisher.getRevenue();
for(Book book: publisher.getBooks()) {
books.add(new BookResponse(book));
}
}
@Data
@NoArgsConstructor
static class BookResponse{
private Long bookId;
private String bookTitle;
private String author;
private String yearPublished;
private String isbn;
private Set<String> genres = new HashSet<String>();
BookResponse(Book book){
bookId = book.getBookId();
bookTitle = book.getBookTitle();
```

```
author = book.getBookAuthor();
yearPublished = book.getYearPublished();
isbn = book.getIsbn();
for(Genre genre: book.getGenres()) {
genres.add(genre.getName());
}
}
}
}
```

BookDao

```
package bookstore.dao;

import org.springframework.data.jpa.repository.JpaRepository;

import bookstore.entity.Book;

public interface BookDao extends JpaRepository<Book, Long> {

}
```

GenreDao

```
package bookstore.dao;

import java.util.Set;

import org.springframework.data.jpa.repository.JpaRepository;

import bookstore.entity.Genre;

public interface GenreDao extends JpaRepository<Genre, Long> {

Set<Genre> findAllByNameIn(Set<String> genres);


}
```

PublisherDao

```
package bookstore.dao;
```

```
import org.springframework.data.jpa.repository.JpaRepository;

import bookstore.entity.Publisher;

public interface PublisherDao extends JpaRepository<Publisher, Long> {

}
```

Book

```
package bookstore.entity;

import java.util.HashSet;
import java.util.Set;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.JoinColumn;
import jakarta.persistence.JoinTable;
import jakarta.persistence.ManyToMany;
import jakarta.persistence.ManyToOne;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.ToString;

@Entity
@Data
public class Book {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long bookId;

@EqualsAndHashCode.Exclude
private String bookTitle;
@EqualsAndHashCode.Exclude
private String bookAuthor;

@EqualsAndHashCode.Exclude
private String yearPublished;

@EqualsAndHashCode.Exclude
private String isbn;

//many books have one author
@EqualsAndHashCode.Exclude
```

```
@ToString.Exclude
@ManyToOne(cascade = CascadeType.ALL)
@JoinColumn(name = "publisher_id", nullable = false)
private Publisher publisher;

// We don't want to delete rows out of the author table if a book is deleted
// but we don't want to delete rows out of the join table
@EqualsAndHashCode.Exclude
@ToString.Exclude
@ManyToMany(cascade = CascadeType.PERSIST)
@JoinTable(name = "book_genre",
joinColumns = @JoinColumn(name = "book_id")
,inverseJoinColumns = @JoinColumn(name = "genre_id"))
private Set<Genre> genres = new HashSet<Genre>();

}
```

Genre

```
package bookstore.entity;

import java.util.HashSet;
import java.util.Set;

import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.ManyToMany;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.ToString;

@Data
@Entity
public class Genre {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long genreId;
@Column(unique = true)
private String name;

//one genre has many books
//if we save a genre that has a set of books, it will save the book as well
//if we delete an genre we want to delete all of the associated books
```

```
@ToString.Exclude
@EqualsAndHashCode.Exclude
@ManyToMany(mappedBy = "genres")
private Set<Book> books = new HashSet<Book>();
}
```

Publisher

```
package bookstore.entity;

import java.util.HashSet;
import java.util.Set;

import jakarta.persistence.CascadeType;
import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;
import jakarta.persistence.OneToMany;
import lombok.Data;
import lombok.EqualsAndHashCode;
import lombok.ToString;

@Data
@Entity
public class Publisher {
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Long publisherId;
private String name;
private String parentCompany;
private String country;
private String revenue;
@ToString.Exclude
@EqualsAndHashCode.Exclude
@OneToMany(mappedBy = "publisher", cascade = CascadeType.ALL)
private Set<Book> books = new HashSet<Book>();

}
```

BookStoreService

```
package bookstore.service;

import java.util.HashSet;
```

```
import java.util.List;
import java.util.NoSuchElementException;
import java.util.Objects;
import java.util.Set;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import bookstore.controller.model.BookData;
import bookstore.controller.model.GenreData;
import bookstore.controller.model.PublisherData;
import bookstore.dao.BookDao;
import bookstore.dao.GenreDao;
import bookstore.dao.PublisherDao;
import bookstore.entity.Book;
import bookstore.entity.Genre;
import bookstore.entity.Publisher;

@Service
public class BookStoreService {
@Autowired
private PublisherDao publisherDao;
@Autowired
private BookDao bookDao;
@Autowired
private GenreDao genreDao;

@Transactional(readOnly = false)
public PublisherData savePublisher(PublisherData publisherData) {
Long publisherId = publisherData.getPublisherId();
Publisher publisher = findOrCreatePublisher(publisherId);
setFieldsInPublisher(publisher, publisherData);
return new PublisherData(publisherDao.save(publisher));
}

private void setFieldsInPublisher(Publisher publisher, PublisherData publisherData)
{
publisher.setName(publisherData.getName());
publisher.setParentCompany(publisherData.getParentCompany());
publisher.setCountry(publisherData.getCountry());
publisher.setRevenue(publisherData.getRevenue());
}

private Publisher findOrCreatePublisher(Long publisherId) {
Publisher publisher;
if(Objects.isNull(publisherId)) {
```

```
publisher = new Publisher();
}
else {
publisher = findPublisherById(publisherId);
}
return publisher;
}

private Publisher findPublisherById(Long publisherId) {
return publisherDao.findById(publisherId).orElseThrow(() -> new
NoSuchElementException(
"Publisher with ID = "+ publisherId + " was not found."));
}

@Transactional(readOnly = true)
public List<PublisherData> retrieveAllPublishers() {
// @formatter:off
return publisherDao.findAll()
.stream()
.map(PublisherData::new)
.toList();
// @formatter:on
}
@Transactional(readOnly = true)
public PublisherData retrievePublisherById(Long publisherId) {
Publisher publisher = findPublisherById(publisherId);
return new PublisherData(publisher);
}
@Transactional(readOnly = false)
public void deletePublisherById(Long publisherId) {
Publisher publisher = findPublisherById(publisherId);
publisherDao.delete(publisher);
}

@Transactional(readOnly = false)
public BookData saveBook(Long publisherId, BookData bookData) {
Publisher publisher = findPublisherById(publisherId);
Set<Genre> genres = genreDao.findAllByNameIn(bookData.getGenres());
Book book = findOrCreateBook(bookData.getBookId());
setBookFields(book, bookData);
book.setPublisher(publisher);
publisher.getBooks().add(book);
for (Genre genre : genres) {
genre.getBooks().add(book);
book.getGenres().add(genre);
}
Book dbBook = bookDao.save(book);
```

```
return new BookData(dbBook);
}

private void setBookFields(Book book, BookData bookData) {
book.setBookTitle(bookData.getBookTitle());
book.setBookAuthor(bookData.getBookAuthor());
book.setYearPublished(bookData.getYearPublished());
book.setIsbn(bookData.getIsbn());
}

private Book findOrCreateBook(Long bookId) {
Book book;
if(Objects.isNull(bookId)) {
book= new Book();
}
else {
book = findBookById(bookId);
}
return book;
}

private Book findBookById(Long bookId) {
return bookDao.findById(bookId)
.orElseThrow(() -> new NoSuchElementException("Book with ID= " + bookId + " does not
exist."));
}
@Transactional(readOnly = true)
public BookData retrieveBookById(Long publisherId, Long bookId) {
findPublisherById(publisherId);
Book book = findBookById(bookId);
if(book.getPublisher().getPublisherId() != publisherId) {
throw new IllegalStateException(
"Book with ID = " + bookId + " is not owned by publisher with ID = " + publisherId);
}
return new BookData(book);
}
public void deleteBookById(Long bookId) {
Book book = findBookById(bookId);
bookDao.delete(book);
}

private Genre findGenreById(Long genreId) {
return genreDao.findById(genreId)
.orElseThrow(() -> new NoSuchElementException("Genre with ID= " + genreId + " does
not exist."));
}
@Transactional(readOnly = true)
```

```
public List<Genre> retrieveAllGenres() {
// @formatter:off
return genreDao.findAll();
}

@Transactional(readOnly = false)
public void deleteGenreById(Long genreId) {
Genre genre = findGenreById(genreId);
genreDao.delete(genre);
}
}
```

application.yaml

```
spring:
datasource:
username: book_store
password: book_store
url: jdbc:mysql://localhost:3306/book_store
jpa:
hibernate:
#none tells hibernate not to create the tables
#create tells hibernate to drop existing tables then create new ones
#update tells hibernate to compare the pre-existing schema and update it
#---> with the new one
#create-drop similar to update, but it will drop the tables a program is stopped
#validate tells hibernate to check if the tables exist otherwise throw an exception
ddl-auto: create-drop
show-sql: true
#tells springbook to wait before populating the tables.
defer-datasource-initialization: true
sql:
init:
#never means that sql will not create and populate the tables
#always means that sql will create and populate the tables
mode: always
```

data.sql

```
INSERT INTO genre (name) VALUES('Action and adventure');
INSERT INTO genre (name) VALUES('Art/architecture');
INSERT INTO genre (name) VALUES('Alternate history');
INSERT INTO genre (name) VALUES('Autobiography');
INSERT INTO genre (name) VALUES('Anthology');
INSERT INTO genre (name) VALUES('Biography');
```

```
INSERT INTO genre (name) VALUES('Chick lit');
INSERT INTO genre (name) VALUES('Business/economics');
INSERT INTO genre (name) VALUES('Childrens');
INSERT INTO genre (name) VALUES('Crafts/hobbies');
INSERT INTO genre (name) VALUES('Classic');
INSERT INTO genre (name) VALUES('Cookbook');
INSERT INTO genre (name) VALUES('Comic book');
INSERT INTO genre (name) VALUES('Diary');
INSERT INTO genre (name) VALUES('Coming-of-age');
INSERT INTO genre (name) VALUES('Dictionary');
INSERT INTO genre (name) VALUES('Crime');
INSERT INTO genre (name) VALUES('Encyclopedia');
INSERT INTO genre (name) VALUES('Drama');
INSERT INTO genre (name) VALUES('Guide');
INSERT INTO genre (name) VALUES('Fairytale');
INSERT INTO genre (name) VALUES('Health/fitness');
INSERT INTO genre (name) VALUES('Fantasy');
INSERT INTO genre (name) VALUES('History');
INSERT INTO genre (name) VALUES('Graphic novel');
INSERT INTO genre (name) VALUES('Home and garden');
INSERT INTO genre (name) VALUES('Historical fiction');
INSERT INTO genre (name) VALUES('Humor');
INSERT INTO genre (name) VALUES('Horror');
INSERT INTO genre (name) VALUES('Journal');
INSERT INTO genre (name) VALUES('Mystery');
INSERT INTO genre (name) VALUES('Math');
INSERT INTO genre (name) VALUES('Paranormal romance');
INSERT INTO genre (name) VALUES('Memoir');
INSERT INTO genre (name) VALUES('Picture book');
INSERT INTO genre (name) VALUES('Philosophy');
INSERT INTO genre (name) VALUES('Poetry');
INSERT INTO genre (name) VALUES('Prayer');
INSERT INTO genre (name) VALUES('Political thriller');
INSERT INTO genre (name) VALUES('Romance');
INSERT INTO genre (name) VALUES('Textbook');
INSERT INTO genre (name) VALUES('Satire');
INSERT INTO genre (name) VALUES('Science fiction');
INSERT INTO genre (name) VALUES('Review');
INSERT INTO genre (name) VALUES('Short story');
INSERT INTO genre (name) VALUES('Science');
INSERT INTO genre (name) VALUES('Suspense');
INSERT INTO genre (name) VALUES('Self help');
INSERT INTO genre (name) VALUES('Thriller');
INSERT INTO genre (name) VALUES('Sports and leisure');
INSERT INTO genre (name) VALUES('Western');
INSERT INTO genre (name) VALUES('Travel');
INSERT INTO genre (name) VALUES('Young adult');
```

```
INSERT INTO genre (name) VALUES('True crime');


INSERT INTO publisher (name, parent_company, country, revenue) VALUES
('Pearson','Pearson PLC','U.K.','$6,070');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('RELX
Group','Reed Elsevier PLC & Reed Elsevier NV','U.K./U.S./Netherlands','$5,609');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('Thomson
Reuters','The Woodbridge Company Ltd.','Canada','$4,941');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES
('Bertelsmann','Bertelsmann AG','Germany','$4,240');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('Wolters
Kluwer','Wolters Kluwer','Netherlands','$3,994');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('Hachette
Livre','Lagardère','France','$2,735');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('Grupo
Planeta','Grupo Planeta','Spain','$1,974');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('Springer
Nature','Springer Nature','Germany','$1,956');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES
('Scholastic','Scholastic','U.S.','$1,742');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('McGraw-Hill
Education','Apollo Global Management LLC','U.S.','$1,719');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES
('Wiley','Wiley','U.S.','$1,719');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES
('HarperCollins','News Corp','U.S.','$1,636');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('Houghton
Mifflin Harcourt','Apax and Omers Capital Partners','U.S./Canada','$1,461');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES
('Holtzbrinck','Houghton Mifflin Harcourt Co.','U.S./Cayman Islands','$1,408');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES
('Informa','Verlagsgruppe Georg von Holtzbrinck','Germany','$1,403');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('Oxford
University Press','Informa PLC','U.K.','$1,229');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES
('Kodansha','Oxford University','U.K.','$1,139');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES
('Shueisha','Kodansha Ltd.','Japan','$1,045');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('Kadokawa
Publishing','Hitotsubashi Group','Japan','$1,041');
INSERT INTO publisher (name, parent_company, country, revenue) VALUES ('Cengage
Learning Holdings II','Kadokawa Holdings Inc.','Japan','$998');
```

URLS:

Git: https://github.com/fmd5045/PromineoFinalProjectBookStore
YouTube: https://youtu.be/EZ8PD7FeyTE