

LU03c - Klassen und Tabellen synchronisieren

Ausgangslage

In der vorhergehenden Lektion haben Sie die Klasse Book aufgrund von der Tabelle books implementiert. Bei einer Änderung müssen Sie dadurch immer sowohl die Tabelle als auch die Klasse anpassen, da diese immer aufeinanderpassen müssen.

SQL books	Java Book
	Book -id: int -title: String -author: String -isbn: String -publicationYear: int +Book(id: int, isbn: String, title: String, author: String, publicationYear: int) +getId(): int +getTitle(): String +setTitle(title: String): void +getAuthor(): String +setAuthor(author: String): void +getIsbn(): String +setIsbn(isbn: String): void +getPublicationYear(): int +setPublicationYear(publicationYear: int): void +toString(): String
publication_year INT	

Um unnötige Arbeitsschritte zu vermeiden, können wir die eine Seite aus der anderen generieren:

- Java-first: Aus den Java-Klassen werden die entsprechenden SQL-Tabellen generiert.
- SQL-first: Aus den SQL-Tabellen werden die entsprechende Java-Klassen generiert.

JPA/Hibernate

Wir wählen den Java-first-Ansatz mit JPA/Hibernate.

Die `build.gradle` wurde im Commit TODO bereits mit den passenden Abhängigkeiten ergänzt. Die zusätzlich Abhängigkeit zu Hikari dient ausserdem, um Datenbankverbindungen zu cachen.

Ebenfalls existiert eine Konfiguration `persistence.xml`. Gewisse Einstellungen wie `jakarta.persistence.jdbc.url`, `jakarta.persistence.jdbc.user` oder `jakarta.persistence.jdbc.password` werden nicht mit der Konfigurationsdatei committed, sondern können wie bis anhin in `config.properties` geschrieben und ersetzen die bisherigen Variablen.

Vorher	Nachher
DB_URL=jdbc:postgresql://localhost:5432/localdb DB_USER=localuser DB_PASSWORD=secret	jakarta.persistence.jdbc.url=jdbc:postgresql://localhost:5432/localdb jakarta.persistence.jdbc.user=localuser jakarta.persistence.jdbc.password=secret

Im Java-Code sind ausserdem folgende Anpassungen nötig:

- Die Eigenschaften der Tabelle werden als Annotationen (@...) in der Java-Klasse gesetzt.
- Ein leerer Standard-Constructor ist nun nötig, damit Hibernate im Hintergrund eine neue Instanz erstellen kann.
- `int` und andere primitive Datentypen werden durch ihre entsprechenden Klassen (`Integer`, ...) ersetzt.

Dadurch sieht die Klasse `Book` neu in etwa so aus. Der Pfad `ch.bzz.model` ist auch in `persistence.xml` hinterlegt und wird in den Tests vorausgesetzt. Verschieben Sie daher die Klasse entsprechend.

```
package ch.bzz.model;

import jakarta.persistence.*;

@Entity
@Table(name = "books")
public class Book {

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

    @Column(name = "isbn", nullable = false, length = 20, unique = true)
    private String isbn;

    @Column(name = "title", nullable = false, length = 255)
    private String title;

    @Column(name = "author", nullable = false, length = 255)
    private String author;

    @Column(name = "publication_year")
    private Integer publicationYear;

    public Book(){}

    public Book(Integer id, String isbn, String title, String author,
Integer publicationYear) {
        this.id = id;
        this.isbn = isbn;
        this.title = title;
        this.author = author;
        this.publicationYear = publicationYear;
    }

    // toString, equals(), hashCode() if needed
    ...
}
```

```
// Getter & Setter
```

```
...
```

Um Objekte von der Datenbank zu lesen kann mittels EntityManager das entsprechende Abfrage ausgeführt werden. Zum Speichern kann persist oder merge (überschreiben falls vorhanden) genutzt werden.

```
public List<Book> getAll(int limit) {
    try (EntityManagerFactory emf =
Persistence.createEntityManagerFactory("localPU", <PROPERTIES>);
        EntityManager em = emf.createEntityManager()) {
        var query = em.createQuery("SELECT b FROM Book b ORDER BY id",
Book.class);
        if (limit > 0) {
            query.setMaxResults(limit);
        }
        return query.getResultList();
    }
}

public void saveAll(List<Book> books) {
    try (EntityManagerFactory emf =
Persistence.createEntityManagerFactory("localPU", <PROPERTIES>);
        EntityManager em = emf.createEntityManager()) {
        try {
            em.getTransaction().begin();
            books.forEach(em::merge);
            em.getTransaction().commit();
        } catch (RuntimeException e) {
            if (em.getTransaction().isActive()) {
                em.getTransaction().rollback();
            }
            log.error("Error during saving of books to the database:",
e);
        }
    }
}
```

From:

<https://wiki.bzz.ch/> - BZZ - Modulwiki

Permanent link:

<https://wiki.bzz.ch/de/modul/ffit/3-jahr/java/learningunits/lu03/c?rev=1756755108>

Last update: 2025/09/01 21:31

