Implementing the Persistence Layer with Spring Boot and JdbcTemplate

1 Schema Overview

For this example:

- Each **Author** can write multiple **Books** (one-to-many relationship).
- We'll use two tables: `Author` and `Book`.

2 Necessary Configurations

Ensure your `application.properties` file has the following configurations:

```
# Datasource configuration
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.datasource.initialization-mode=always

# Enable SQL logging
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
```

Listing 1: application.properties

Optionally, you can add an SQL script in `src/main/resources/schema.sql` to create tables on startup.

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

CREATE TABLE Book (
    id BIGINT PRIMARY KEY AUTO_INCREMENT,
    title VARCHAR(100) NOT NULL,
    author_id BIGINT,
    FOREIGN KEY (author_id) REFERENCES Author(id)
);
```

Listing 2: schema.sql

You can add sample data in `data.sql`:

```sql
INSERT INTO Author (name) VALUES ('Author 1'), ('Author 2');
INSERT INTO Book (title, author_id) VALUES ('Book 1', 1), ('Book 2', 1), ('Book 3', 2);
```

Listing 3: data.sql

3 Architecture for Persistence Layer with `JdbcTemplate`

The architecture will consist of:

- **Data Transfer Objects (DTOs)**: Simple classes for `Author` and `Book`.
- **DAO Interface**: Defines CRUD operations for `Author` and `Book`.
- **DAO Implementation**: Uses `JdbcTemplate` for database operations.

4 Step-by-Step Implementation

4.1 Create Data Transfer Objects (DTOs)

```java
public class Author {
    private Long id;
    private String name;

    // Constructors, getters, setters
    public Author(Long id, String name) {
        this.id = id;
        this.name = name;
    }

    public Author() {}

    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; }
}
```

Listing 4: Author.java

```java
public class Book {
    private Long id;
    private String title;
    private Long authorId;

    // Constructors, getters, setters
    public Book(Long id, String title, Long authorId) {
        this.id = id;
        this.title = title;
        this.authorId = authorId;
    }

    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 Long getAuthorId() { return authorId; }
    public void setAuthorId(Long authorId) { this.authorId = authorId; }
}
```

Listing 5: Book.java

4.2 Define DAO Interfaces

```java
import java.util.List;

public interface AuthorDAO {
    Author findById(Long id);
    List<Author> findAll();
    int save(Author author);
    int update(Author author);
    int deleteById(Long id);
}
```

Listing 6: AuthorDAO.java

```java
import java.util.List;

public interface BookDAO {
    Book findById(Long id);
    List<Book> findAllByAuthorId(Long authorId);
    int save(Book book);
    int update(Book book);
    int deleteById(Long id);
}
```

Listing 7: BookDAO.java

4.3 Implement DAO Interfaces with `JdbcTemplate`

```java
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

@Repository
public class AuthorDAOImpl implements AuthorDAO {

    private final JdbcTemplate jdbcTemplate;

    public AuthorDAOImpl(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    private RowMapper<Author> rowMapper = new RowMapper<>() {
        public Author mapRow(ResultSet rs, int rowNum) throws SQLException {
            return new Author(rs.getLong("id"), rs.getString("name"));
        }
    };

    @Override
    public Author findById(Long id) {
        String sql = "SELECT * FROM Author WHERE id = ?";
        return jdbcTemplate.queryForObject(sql, rowMapper, id);
    }

    @Override
    public List<Author> findAll() {
        String sql = "SELECT * FROM Author";
        return jdbcTemplate.query(sql, rowMapper);
    }

    @Override
    public int save(Author author) {
        String sql = "INSERT INTO Author (name) VALUES (?)";
        return jdbcTemplate.update(sql, author.getName());
    }

    @Override
    public int update(Author author) {
        String sql = "UPDATE Author SET name = ? WHERE id = ?";
        return jdbcTemplate.update(sql, author.getName(), author.getId());
    }

    @Override
    public int deleteById(Long id) {
        String sql = "DELETE FROM Author WHERE id = ?";
        return jdbcTemplate.update(sql, id);
    }
}
```

Listing 8: AuthorDAOImpl.java

```java
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.RowMapper;
import org.springframework.stereotype.Repository;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

@Repository
public class BookDAOImpl implements BookDAO {

    private final JdbcTemplate jdbcTemplate;

    public BookDAOImpl(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    private RowMapper<Book> rowMapper = new RowMapper<>() {
        public Book mapRow(ResultSet rs, int rowNum) throws SQLException {
            return new Book(rs.getLong("id"), rs.getString("title"), rs.getLong("author_id"));
        }
    };

    @Override
    public Book findById(Long id) {
        String sql = "SELECT * FROM Book WHERE id = ?";
        return jdbcTemplate.queryForObject(sql, rowMapper, id);
    }

    @Override
    public List<Book> findAllByAuthorId(Long authorId) {
        String sql = "SELECT * FROM Book WHERE author_id = ?";
        return jdbcTemplate.query(sql, rowMapper, authorId);
    }

    @Override
    public int save(Book book) {
        String sql = "INSERT INTO Book (title, author_id) VALUES (?, ?)";
        return jdbcTemplate.update(sql, book.getTitle(), book.getAuthorId());
    }

    @Override
    public int update(Book book) {
        String sql = "UPDATE Book SET title = ?, author_id = ? WHERE id = ?";
        return jdbcTemplate.update(sql, book.getTitle(), book.getAuthorId(), book.getId());
    }

    @Override
    public int deleteById(Long id) {
        String sql = "DELETE FROM Book WHERE id = ?";
        return jdbcTemplate.update(sql, id);
    }
}
```

Listing 9: BookDAOImpl.java

5 Summary

This architecture separates data access logic into DAOs, with `JdbcTemplate` handling SQL interactions. It is simple, efficient, and allows easy maintenance and testing.