Library DAO Tutorial

Step 5: JdbcTemplate Version – DAO Implementation

The Learning House

427 S 4th Street #300

Louisville KY 40202

Step 5: JdbcTemplate Version – DAO Implementation

Overview

In this step, we will implement the JdbcTemplate version of our DAO. This DAO will be similar to other DAOs you have implemented in class (e.g., prepared statements, JdbcTemplate calls, mapper objects), but the one-to-many and many-to-many relationships will add some complexity. You will notice that the methods for the Book objects all require a two-step process: one step operates on the books table and the other operates on the books_authors table.

Implementation

Notes:

- Notice that all of the book related operations are two-step processes: one step deals with the books table, the other deals with the books_authors table
- Notice the helper methods for book related operations.

```
package com.swcguild.library.dao;

import com.swcguild.library.model.Author;
import com.swcguild.library.model.Book;
import com.swcguild.library.model.Publisher;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.time.LocalDate;
import java.util.List;
import org.springframework.dao.EmptyResultDataAccessException;
import org.springframework.jdbc.core.BatchPreparedStatementSetter;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.core.simple.ParameterizedRowMapper;
import org.springframework.transaction.annotation.Propagation;
import org.springframework.transaction.annotation.Transactional;

/**
 *
 * @author apprentice
 */
public class LibraryDaoDbImpl implements LibraryDao {

    private JdbcTemplate jdbcTemplate;

    public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
        this.jdbcTemplate = jdbcTemplate;
    }

    // AUTHORS
    private static final String SQL_INSERT_AUTHOR
           = "insert into authors (first_name, last_name, street, city, state, zip, phone) values (?, ?, ?, ?, ?, ?, ?)";
    private static final String SQL_DELETE_AUTHOR
            = "delete from authors where author_id = ?";
    private static final String SQL_UPDATE_AUTHOR
            = "update authors set first_name = ?, last_name = ?, street = ?, city = ?, state = ?, zip = ?, phone
= ? where author_id = ?";
    private static final String SQL_SELECT_AUTHOR
            = "select * from authors where author_id = ?";
```

```
    private static final String SQL_SELECT_AUTHORS_BY_BOOK_ID
            = "select au.author_id, au.first_name, au.last_name, au.street, au.city, au.state, au.zip, au.phone
from authors au "
            + "join books_authors ba on au.author_id = ba.author_id where ba.book_id = ?";
    private static final String SQL_SELECT_ALL_AUTHORS
            = "select * from authors";

    // BOOKS AND BOOKS_AUTHORS
    private static final String SQL_INSERT_BOOK
            = "insert into books (isbn, title, publisher_id, price, publish_date) values (?, ?, ?, ?, ?)";
    private static final String SQL_INSERT_BOOKS_AUTHORS
            = "insert into books_authors (book_id, author_id) values(?, ?)";
    private static final String SQL_DELETE_BOOK
            = "delete from books where book_id = ?";
    private static final String SQL_DELETE_BOOKS_AUTHORS
            = "delete from books_authors where book_id = ?";
    private static final String SQL_UPDATE_BOOK
            = "update books set isbn = ?, title = ?, publisher_id = ?, price = ?, publish_date = ? where book_id
= ?";
    private static final String SQL_SELECT_BOOK
            = "select * from books where book_id = ?";
    private static final String SQL_SELECT_BOOKS_AUTHORS_AUTHOR_ID_BY_BOOK_ID
            = "select author_id from books_authors where book_id = ?";
    private static final String SQL_SELECT_ALL_BOOKS
            = "select * from books";
    private static final String SQL_SELECT_BOOKS_BY_AUTHOR_ID
            = "select b.book_id, b.isbn, b.title, b.publisher_id, b.price, b.publish_date from books b join
books_authors ba on author_id where b.book_id = ba.book_id and ba.author_id = ?;";
    private static final String SQL_SELECT_BOOKS_BY_PUBLISHER_ID
            = "select * from books where publisher_id = ?";

    // PUBLISHERS
    private static final String SQL_INSERT_PUBLISHER
            = "insert into publishers (name, street, city, state, zip, phone) values (?, ?, ?, ?, ?, ?)";
    private static final String SQL_DELETE_PUBLISHER
            = "delete from publishers where publisher_id = ?";
    private static final String SQL_UPDATE_PUBLISHER
            = "update publishers set name = ?, street = ?, city = ?, state = ?, zip = ?, phone = ? where
publisher_id = ?";
    private static final String SQL_SELECT_PUBLISHER
            = "select * from publishers where publisher_id = ?";
    private static final String SQL_SELECT_PUBLISHER_BY_BOOK_ID
            = "select pub.publisher_id, pub.name, pub.street, pub.city, pub.state, pub.zip, pub.phone from
publishers pub "
            + "join books on pub.publisher_id = books.publisher_id where books.book_id = ?";
    private static final String SQL_SELECT_ALL_PUBLISHERS
            = "select * from publishers";

    // AUTHOR METHODS
    // ==============
    @Override
    @Transactional(propagation = Propagation.REQUIRED, readOnly = false)
    public void addAuthor(Author author) {
        jdbcTemplate.update(SQL_INSERT_AUTHOR,
                author.getFirstName(),
                author.getLastName(),
                author.getStreet(),
                author.getCity(),
                author.getState(),
                author.getZip(),
                author.getPhone());
        author.setAuthorId(jdbcTemplate.queryForObject("select LAST_INSERT_ID()", Integer.class));
```

```
    }

    @Override
    public void deleteAuthor(int authorId) {
        jdbcTemplate.update(SQL_DELETE_AUTHOR, authorId);
    }

    @Override
    public void updateAuthor(Author author) {
        jdbcTemplate.update(SQL_UPDATE_AUTHOR,
                author.getFirstName(),
                author.getLastName(),
                author.getStreet(),
                author.getCity(),
                author.getState(),
                author.getZip(),
                author.getPhone(),
                author.getAuthorId());
    }

    @Override
    public Author getAuthorById(int authorId) {
        try {
            return jdbcTemplate.queryForObject(SQL_SELECT_AUTHOR, new AuthorMapper(), authorId);
        } catch (EmptyResultDataAccessException ex) {
            return null;
        }
    }

    @Override
    public List<Author> getAuthorsByBookId(int bookId) {
        return jdbcTemplate.query(SQL_SELECT_AUTHORS_BY_BOOK_ID, new AuthorMapper(), bookId);
    }

    @Override
    public List<Author> getAllAuthors() {
        return jdbcTemplate.query(SQL_SELECT_ALL_AUTHORS, new AuthorMapper());
    }

    // BOOK METHODS
    // ============
    @Override
    @Transactional(propagation = Propagation.REQUIRED, readOnly = false)
    public void addBook(Book book) {
        // first insert into books table and get newly generated book_id
        jdbcTemplate.update(SQL_INSERT_BOOK,
                book.getIsbn(),
                book.getTitle(),
                book.getPublisherId(),
                book.getPrice(),
                book.getPublishDate().toString());
        book.setBookId(jdbcTemplate.queryForObject("select LAST_INSERT_ID()", Integer.class));
        // now update the books_authors table
        insertBooksAuthors(book);
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRED, readOnly = false)
    public void deleteBook(int bookId) {
        // delete books_authors relationship for this book
        jdbcTemplate.update(SQL_DELETE_BOOKS_AUTHORS, bookId);
        // delete book
```

```
        jdbcTemplate.update(SQL_DELETE_BOOK, bookId);
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRED, readOnly = false)
    public void updateBook(Book book) {
        // update books table
        jdbcTemplate.update(SQL_UPDATE_BOOK,
                book.getIsbn(),
                book.getTitle(),
                book.getPublisherId(),
                book.getPrice(),
                book.getPublishDate().toString(),
                book.getBookId());

        // delete books_authors relationships and then reset them
        jdbcTemplate.update(SQL_DELETE_BOOKS_AUTHORS, book.getBookId());
        insertBooksAuthors(book);
    }

    @Override
    @Transactional(propagation = Propagation.REQUIRED, readOnly = true)
    public Book getBookById(int id) {
        try {
            // get the properties from the books table
            Book b = jdbcTemplate.queryForObject(SQL_SELECT_BOOK, new BookMapper(), id);
            // get the author ids for this book
            int[] idArray = getAuthorIdsForBook(b);
            // set the author ids on the book
            b.setAuthorIds(idArray);
            return b;
        } catch (EmptyResultDataAccessException ex) {
            return null;
        }
    }

    @Override
    public List<Book> getBooksByAuthorId(int authorId) {
        // get the books written/cowritten by this author
        List<Book> bList = jdbcTemplate.query(SQL_SELECT_BOOKS_BY_AUTHOR_ID, new BookMapper(), authorId);
        // set the complete list of author ids for each book
        for (Book b : bList) {
            b.setAuthorIds(getAuthorIdsForBook(b));
        }
        return bList;
    }

    @Override
    public List<Book> getBooksByPublisherId(int publisherId) {
        // get the books published by this publishers
        List<Book> bList = jdbcTemplate.query(SQL_SELECT_BOOKS_BY_PUBLISHER_ID, new BookMapper(), publisherId);
        // set the complete list of author ids for each book
        for (Book b : bList) {
            b.setAuthorIds(getAuthorIdsForBook(b));
        }
        return bList;
    }

    @Override
    public List<Book> getAllBooks() {
        // get all the books
        List<Book> bList = jdbcTemplate.query(SQL_SELECT_ALL_BOOKS, new BookMapper());
```

```
        // set the complete list of author ids for each book
        for (Book b : bList) {
            b.setAuthorIds(getAuthorIdsForBook(b));
        }
        return bList;
    }

    // PUBLISHER METHODS
    // =================
    @Override
    @Transactional(propagation = Propagation.REQUIRED, readOnly = false)
    public void addPublisher(Publisher publisher) {
        jdbcTemplate.update(SQL_INSERT_PUBLISHER,
                publisher.getName(),
                publisher.getStreet(),
                publisher.getCity(),
                publisher.getState(),
                publisher.getZip(),
                publisher.getPhone());
        publisher.setPublisherId(jdbcTemplate.queryForObject("select LAST_INSERT_ID()",
                                                             Integer.class));
    }

    @Override
    public void deletePublisher(int publisherId) {
        jdbcTemplate.update(SQL_DELETE_PUBLISHER, publisherId);
    }

    @Override
    public void updatePublisher(Publisher publisher) {
        jdbcTemplate.update(SQL_UPDATE_PUBLISHER,
                publisher.getName(),
                publisher.getStreet(),
                publisher.getCity(),
                publisher.getState(),
                publisher.getZip(),
                publisher.getPhone(),
                publisher.getPublisherId());
    }

    @Override
    public Publisher getPublisherById(int id) {
        try {
            return jdbcTemplate.queryForObject(SQL_SELECT_PUBLISHER,
                                               new PublisherMapper(),
                                               id);
        } catch (EmptyResultDataAccessException ex) {
            return null;
        }
    }

    @Override
    public Publisher getPublisherByBookId(int bookId) {
        return jdbcTemplate.queryForObject(SQL_SELECT_PUBLISHER_BY_BOOK_ID,
                                           new PublisherMapper(),
                                           bookId);
    }

    @Override
    public List<Publisher> getAllPublishers() {
        return jdbcTemplate.query(SQL_SELECT_ALL_PUBLISHERS, new PublisherMapper());
    }
```

```
    // HELPERS
    // =======
    /**
     * Inserts authors for this book into the books_authors table
     * @param book
     */
    private void insertBooksAuthors(Book book) {
        final int bookId = book.getBookId();
        final int[] authorIds = book.getAuthorIds();
        // use the batchUpdate so we only make one call to the database
        jdbcTemplate.batchUpdate(SQL_INSERT_BOOKS_AUTHORS, new BatchPreparedStatementSetter() {
            @Override
            public void setValues(PreparedStatement ps, int i) throws SQLException {
                ps.setInt(1, bookId);
                ps.setInt(2, authorIds[i]);
            }

            @Override
            public int getBatchSize() {
                return authorIds.length;
            }
        });
    }

    /**
     * Gets the array of author ids associated with the given book
     * @param book
     * @return
     */
    private int[] getAuthorIdsForBook(Book book) {
        // get the list of authors ids from the books_authors table
        List<Integer> authorIds = jdbcTemplate.queryForList(SQL_SELECT_BOOKS_AUTHORS_AUTHOR_ID_BY_BOOK_ID, new
Integer[]{book.getBookId()}, Integer.class);
        //
        int[] idArray = new int[authorIds.size()];
        for (int i = 0; i < authorIds.size(); i++) {
            idArray[i] = authorIds.get(i);
        }
        return idArray;
    }

    // MAPPERS
    // =======
    private static final class AuthorMapper implements ParameterizedRowMapper<Author> {

        @Override
        public Author mapRow(ResultSet rs, int i) throws SQLException {
            Author au = new Author();
            au.setFirstName(rs.getString("first_name"));
            au.setLastName(rs.getString("last_name"));
            au.setStreet(rs.getString("street"));
            au.setCity(rs.getString("city"));
            au.setState(rs.getString("state"));
            au.setZip(rs.getString("zip"));
            au.setPhone(rs.getString("phone"));
            au.setAuthorId(rs.getInt("author_id"));
            return au;
        }
    }

    private static final class PublisherMapper implements ParameterizedRowMapper<Publisher> {
```

```
        @Override
        public Publisher mapRow(ResultSet rs, int i) throws SQLException {
            Publisher pub = new Publisher();
            pub.setPublisherId(rs.getInt("publisher_id"));
            pub.setName(rs.getString("name"));
            pub.setStreet(rs.getString("street"));
            pub.setCity(rs.getString("city"));
            pub.setState(rs.getString("state"));
            pub.setZip(rs.getString("zip"));
            pub.setPhone(rs.getString("phone"));
            return pub;
        }
    }

    private static final class BookMapper implements ParameterizedRowMapper<Book> {

        @Override
        public Book mapRow(ResultSet rs, int i) throws SQLException {
            Book b = new Book();
            b.setBookId(rs.getInt("book_id"));
            b.setIsbn(rs.getString("isbn"));
            b.setTitle(rs.getString("title"));
            b.setPublisherId(rs.getInt("publisher_id"));
            b.setPrice(rs.getBigDecimal("price"));
            b.setPublishDate(rs.getTimestamp("publish_date").toLocalDateTime().toLocalDate());

            return b;
        }
    }
}
```

Spring Configuration File

Now that we have all the code in place, we need to configure Spring for the following:

1. MySQL datasource and connection pooling
2. Transactions:
    a. Annotation driven transactions
    b. Transaction Manager
3. JdbcTemplate (wiring in the datasource defined in step 1)
4. DAO - bean definition for our DAO implementation (wiring in the JdbcTemplate defined in step 3)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:tx="http://www.springframework.org/schema/tx"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/mvc
  http://www.springframework.org/schema/mvc/spring-mvc.xsd
```

```
  http://www.springframework.org/schema/tx
  http://www.springframework.org/schema/tx/spring-tx-3.0.xsd
  http://www.springframework.org/schema/context
  http://www.springframework.org/schema/context/spring-context.xsd
  http://www.springframework.org/schema/aop
http://www.springframework.org/schema/aop/spring-aop-3.2.xsd">

    <!-- Bean definitions go here -->
    <tx:annotation-driven/>

    <context:component-scan base-package="com.swcguild.library.dao" />

    <bean id="dataSource"
          class="org.apache.commons.dbcp.BasicDataSource">
        <property name="driverClassName" value="com.mysql.jdbc.Driver"/>
        <property name="url"
                  value="jdbc:mysql://localhost:3306/library_test"/>
        <property name="username" value="root"/>
        <property name="password" value="root"/>
        <property name="initialSize" value="5"/>
        <property name="maxActive" value="10"/>
    </bean>

    <bean id="transactionManager"
          class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="jdbcTemplate"
          class="org.springframework.jdbc.core.JdbcTemplate">
        <property name="dataSource" ref="dataSource"/>
    </bean>

    <bean id="libraryDao" class="com.swcguild.library.dao.LibraryDaoDbImpl">
        <property name="jdbcTemplate" ref="jdbcTemplate"/>
    </bean>
</beans>
```

Wrap-up

In this step, we implemented the JdbcTemplate-based DAO and examined how the one-to-many and many-to-many relationships in the database affect our Java code. In the next step, we will begin the Hibernate implementation of our DAO.