Mastering the Spring Framework

Chapter 6: Database operations with Spring Boot

Chapter Objectives

In this chapter, we will:

- H2 Database
- Jdbc Template
- JPA

Spring Boot H2 Database

- **In-Memory Database**
  - Relies on system memory as oppose to disk space for storage of data
  - Databases are volatile, by default, and all stored data loss when we restart the application
  - Widely used in-memory databases are **H2, HSQLDB** (HyperSQL Database)**,** and **Apache Derby**
- **H2 Database**
  - **H2** is an **embedded, open-source,** and **in-memory** database
  - **Advantages**
    - Zero configuration
    - It is easy to use.
    - It is lightweight and fast.
    - It supports standard SQL and JDBC API.
    - It provides a web console to maintain in the database.

Configure H2 Database

- We need to add the following dependency in pom.xml file

```
<dependency>
  <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <scope>runtime</scope>
</dependency>
```

- After adding the dependency, we need to configure **data source URL, driver class name, username,** and **password** of H2 database in **application.properties** file.

```
spring.datasource.url=jdbc:h2:file:~/test

spring.datasource.username=sa

spring.datasource.password=

spring.datasource.driver-class-name=org.h2.Driver
```

Configure H2 Database (Continued)

- Create Schema & Populate Data
  - We can create schema and populate data by creating a SQL files in the **resource** folder (src/main/resource).
  - Spring Boot automatically picks up the **schema.sql** & **data.sql** files and run it against the H2 database during the application start-up.
- H2 Console
  - By default, the console view of the H2 database is disabled.
  - Before accessing the H2 database, we must enable it by using the following property and setup the url where we can access it.

```
spring.h2.console.enabled=true

spring.h2.console.path=/h2
```

Jdbc vs jdbcTemplate

- JDBC produces a lot of boiler plate code, such as
  - opening/closing a connection to a database,
  - handling sql exceptions etc.
  - It makes the code extremely cumbersome and difficult to read.
- Spring provides a simplification in handling database access with the Spring JDBC Template.
  - The JdbcTemplate class executes SQL queries,
  - iterates over the ResultSet, and retrieves the called values,
  - "catches" the exceptions, and translates them into the exceptions
  - Instances of the JdbcTemplate class are thread-safe. This means that by configuring a single instance of the JdbcTemplate class, we can then use it for several DAO objects.

Methods in jdbcTemplate

- public List query(String sql, RowMapper rse)
  - used to fetch records using RowMapper.
- public List queryForObject(String sql, new Object, RowMapper rse)
  - to **fetch** a single row record from database, and convert the row into an object via row mapper
- public int update(String query)
  - used to insert, update and delete records.
- public void execute(String query)
  - used to execute DDL query.

RowMapper

- RowMapper maps rows of a result set on a per-row basis.
- Implementations of this interface perform the actual work of mapping each row to a result object.

```
(rs, rowNum) ->
        new Book(
                rs.getLong("id"),
                rs.getString("name"),
                rs.getBigDecimal("price")
        )
```

Using named parameters

- NamedParameterJdbcTemplate is a template class with a basic set of JDBC operations, allowing the use of named parameters rather than traditional '?' placeholders.

Spring Data JPA

- Spring Data JPA provides complete abstraction over the DAO layer.
  - We don't need to write the implementation for the DAO layer anymore;
  - Spring Data auto-generates the implementation DAO implementations.

EntityBean

- The first code level thing we will do is write an Entity Bean.
- Here is what the Oracle Documentation says about Entity Beans.
  - Using JPA, you can designate any POJO class as a JPA entity – a Java object whose nontransient fields should be persisted to a relational database using the services of an entity manager obtained from a JPA persistence provider
  - In simpler words, the JPA Entity is any Java POJO, which can represent the underlying table structure.

EntityBean (Continued)

- @Entity
  - The above POJO is annotated with @Entity
    - which is to denote this is an entity object for the table name with the same name as the POJO.
- @Id
  - Denotes Primary Key
- @GeneratedValue
  - Denotes that the primary key is auto generated

```
@Entity
public class Books {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private BigDecimal price;
```

Repository Interface

- The Repository represents the DAO layer, which typically does all the database operations.
- The Spring Data Repositories need to know about the the Entity and Id fields they are going to deal with.
  - This information is provided in the repository declaration.
  - Usually each Entity will have its own dedicated Repository.

Named Queries (@Queries)

- Sometimes, the query methods do not fulfil the requirements of all types of queries.
- In these cases too, we can still use Spring repositories and provide queries with *@Query*.
- Consider we are querying an *Event* table and wish to find the largest *event_id*.
  - Below is a way we can use @Query to provide a query.
  - Thing to remember: *method name doesn't have any significance here.*

```
@Query(" SELECT MAX(eventId) AS eventId FROM Event ")
Long lastProcessedEvent();
```

Repository Interface (Continued)

- Spring knows the *Books* is the entity and *Long* is the type of primary key.
- *BookRepository* inherits all the methods from CrudRepository.
- Additionally, it defines a custom query method findByName.
- @Repository annotates it as a Repository.
- Spring provides proxy implementation of *BookRepository* and all of its methods including *findByName*.

```
public interface BookRepository extends CrudRepository<Books, Long> {

    List<Books> findByName(String name);

}
```

Spring Boot CrudRepository

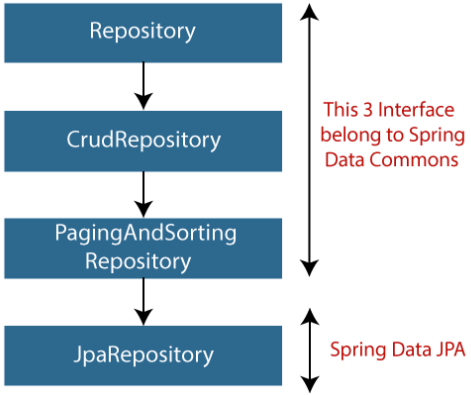
- Spring Boot provides an interface called **CrudRepository** that contains methods for CRUD operations.
  - If we want to use CrudRepository in an application, we have to create an interface and extend the **CrudRepository**.
- Example:

```
public interface StudentRepository extends CrudRepository<Student, Integer> {}
```

  - In the above example, we have created an interface named **StudentRepository** that extends CrudRepository.
  - Where **Student** is the repository to manage, and **Integer** is the type of Id that is defined in the Student repository.

Spring Boot JpaRepository

- JpaRepository provides JPA related methods such as flushing, persistence context, and deletes a record in a batch.
  - JpaRepository extends both **CrudRepository** and **PagingAndSortingRepository.**
- Example: **public interface** BookDAO **extends** JpaRepository {}
- Spring Data Repository Interface



CrudRepository

```
public interface CrudRepository<T, ID>
        extends Repository<T, ID> {

    <S extends T> S save(S var1);

    <S extends T> Iterable<S> saveAll(Iterable<S> var1);

    Optional<T> findById(ID var1);

    boolean existsById(ID var1);

    Iterable<T> findAll();

    Iterable<T> findAllById(Iterable<ID> var1);

    long count();

    void deleteById(ID var1);

    void delete(T var1);

    void deleteAll(Iterable<? extends T> var1);

    void deleteAll();
}
```

Chapter Summary

In this chapter, we have:

- H2 Database
- Jdbc Template
- JPA