In Spring Boot JDBC, the database related beans like DataSource, JdbcTemplate and NamedParameterJdbcTemplate will be configured and created during the startup, to use it, just @Autowired the bean you want, for examples:

@Autowired

JdbcTemplate jdbcTemplate;

@Autowired

private NamedParameterJdbcTemplate jdbcTemplate;

To connect to a database (e.g MySQL), include the JDBC driver in the project classpath:

pom.xml

<!-- MySQL JDBC driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-jdbc</artifactId>

</dependency>

And define the datasoure properties in application.properties

application.properties

## MySQL

#spring.datasource.url=jdbc:mysql://192.168.1.4:3306/test

#spring.datasource.username=mkyong

#spring.datasource.password=password

# Oracle

#spring.datasource.url=jdbc:oracle:thin:@localhost:1521:orcl

#spring.datasource.username=system

#spring.datasource.password=Password123

@Repository

public class JdbcBookRepository implements BookRepository {

// Spring Boot will create and configure DataSource and JdbcTemplate

// To use it, just @Autowired

@Autowired

private JdbcTemplate jdbcTemplate;

@Override

public int count() {

return jdbcTemplate

.queryForObject("select count(\*) from books", Integer.class);

}

@Override

public int save(Book book) {

return jdbcTemplate.update(

"insert into books (name, price) values(?,?)",

book.getName(), book.getPrice());

}

@Override

public int update(Book book) {

return jdbcTemplate.update(

"update books set price = ? where id = ?",

book.getPrice(), book.getId());

}

@Override

public int deleteById(Long id) {

return jdbcTemplate.update(

"delete books where id = ?",

id);

}

@Override

public List<Book> findAll() {

return jdbcTemplate.query(

"select \* from books",

(rs, rowNum) ->

new Book(

rs.getLong("id"),

rs.getString("name"),

rs.getBigDecimal("price")

)

);

}

// jdbcTemplate.queryForObject, populates a single object

@Override

public Optional<Book> findById(Long id) {

return jdbcTemplate.queryForObject(

"select \* from books where id = ?",

new Object[]{id},

(rs, rowNum) ->

Optional.of(new Book(

rs.getLong("id"),

rs.getString("name"),

rs.getBigDecimal("price")

))

);

}

@Override

public List<Book> findByNameAndPrice(String name, BigDecimal price) {

return jdbcTemplate.query(

"select \* from books where name like ? and price <= ?",

new Object[]{"%" + name + "%", price},

(rs, rowNum) ->

new Book(

rs.getLong("id"),

rs.getString("name"),

rs.getBigDecimal("price")

)

);

}

@Override

public String getNameById(Long id) {

return jdbcTemplate.queryForObject(

"select name from books where id = ?",

new Object[]{id},

String.class

);

}

}