AIF233201 - Pemrograman Berbasis Web

Database

by Raymond Chandra Putra
Pascal Alfadian Nugroho
Keenan Adiwijaya Leman

31-10-2024

INFORMATIKA
UNPAR

Overview

- Database
- JDBC
- JDBC Template
- Repository
- Spring Data

Database

INFORMATIKA
UNPAR

Database and Web Application

- We use database to store information.
- The storage must be persistent (non-volatile).
  - We shouldn’t relied on our server stability, and risk our user data.
- The data is large, using our the server RAM to store persistent information isn’t practical nor economical.

SQL-Based (RDBMS)

- We use RDBMS when our data is relational in nature.
- SQL-based database technology is mature.
- For example:
  - PostgreSQL: feature rich, and complex.
  - Microsoft SQL: feature rich, and complex. (commercial)
  - MySQL: (generally) less feature, less complex.

No-SQL Database

- When we have unstructured data, No-SQL is more preferred.
- (Generally) faster than RDBMS.
- Far more flexible.
- More scalable (horizontally).
- Example:
  - MongoDB
  - Redis
  - Firebase

PostgreSQL

- This course use PostgreSQL as DBMS choice.
- PostgreSQL is an object-relational DBMS based on POSTGRES, Version 4.2, developed at the University of California at Berkeley Computer Science Department.
- POSTGRES pioneered many concepts that only became available in some commercial database systems much later.
- It supports a large part of the SQL standard and offers many modern features.

Read more: https://www.postgresql.org/about/

TODO:

- Please install PostgreSQL (server) and pgAdmin 4 (or preferred client) in your environment.
- Please explore how to use, query, see the result, and differentiate between postgreSQL with other(s) DBMS query you've learned before.

DataSource

- The DataSource works as a factory for providing database connections. A datasource uses a URL along with username/password credentials to establish the database connection.
- In Java, a datasource implements the `javax.sql.DataSource` interface.
- We may use a datasource to obtain standard `Connection` object.

Implementation

- Create DataSource Bean

```
8   @Configuration
9   public class DataSourceConfig {
10
11      @Value("${spring.datasource.url}")
12      private String url;
13
14      @Value("${spring.datasource.username}")
15      private String username;
16
17      @Value("${spring.datasource.password}")
18      private String password;
19
20      @Bean
21      public DataSource dataSource() {
22          DriverManagerDataSource dataSource = new DriverManagerDataSource();
23          dataSource.setUrl(url);
24          dataSource.setUsername(username);
25          dataSource.setPassword(password);
26          return dataSource;
27      }
28  }
```

Implementation (2)

- @Value get the data from application.properties

```
3   # Postgre-specific properties
4   spring.datasource.url=jdbc:postgresql://localhost:5432/pbw
5   spring.datasource.username=postgres
6   spring.datasource.password=postgres
7
8   # HikariCP-specific properties
9   spring.datasource.hikari.maximum-pool-size=10
10  spring.datasource.hikari.minimum-idle=5
11  spring.datasource.hikari.idle-timeout=30000
```

*change value according to your environment.

HikariCP

- HikariCP is a high-performance JDBC connection pool. It is one of the most popular connection pooling libraries used in Java applications for managing database connections efficiently.
- A connection pool is a collection of reusable connections to a database that are maintained by the application, rather than opening and closing new connections for each database interaction.
- This improves performance and reduces the overhead of repeatedly establishing connections to the database.
- HikariCP automatically added when we use `spring-boot-starter-jdbc.`

The Basic of Querying
(common case)

The process of using SQL with web application:

1. Connect to DBMS.
2. Select the database to use.
3. Build a query string.
4. Perform the query.
5. Retrieve the results and output them to a web page (if needed).
6. Repeat Steps 3 through 5 until all desired data has been retrieved.
7. Disconnect from database.

Starter JDBC

- To query the database using JDBC we need to include the JDBC package in build.gradle.kts:
  - `implementation("org.springframework.boot:spring-boot-starter-jdbc")`
  - `runtimeOnly("org.postgresql:postgresql")`
- In this slide we'll give you 2 way of accessing database:
  - Basic JDBC (Low-level JDBC)
  - JDBC Template (High-level abstraction)

Basic JDBC - Implementation

```
27      @Autowired
28      private DataSource dataSource;
29
30      public Optional<Person> findByIdNonPrep(int id) {
31          Connection connection = null;
32          Statement statement = null;
33          ResultSet resultSet = null;
34          try {
35              connection = dataSource.getConnection();
36              statement = connection.createStatement();
37              String sql = "select id, fname, lname from Person where id="+id;
38              resultSet = statement.executeQuery(sql);
39
40              Person person = null;
41              if(resultSet.next()) {
42                  person = new Person(
43                      resultSet.getInt("id"),
44                      resultSet.getString("fname"),
45                      resultSet.getString("lname")
46                  );
47              }
48              return Optional.of(person);
49          } catch (SQLException e) {
50          }
```

INFORMATIKA UNPAR

Implementation (cont)

```
51      finally {
52          if (resultSet != null) {
53              try {
54                  resultSet.close();
55              } catch (SQLException e) {}
56          }
57          if (statement != null) {
58              try {
59                  statement.close();
60              } catch (SQLException e) {}
61          }
62
63          if (connection != null) {
64              try {
65                  connection.close();
66              } catch (SQLException e) {}
67          }
68      }
69      return Optional.empty();
70  }
```

Implementation (cont)

- Call method from routing:

```
21      @GetMapping("/")
22      public String index(Model model) {
23          Optional<Person> person = this.findByIdNonPrep(1);
24          if(!person.isEmpty()){
25              model.addAttribute("name", person.get().getFullName());
26          }
27          return "index";
28      }
```

Intermezzo: SQL injection

- Example 1:

```
String username = " "; // insert anything
String password = " ' OR 'a'='a ";  // insert ' OR 'a'='a

String sqlQuery = "SELECT * FROM users where username='" +
expected_data + "' AND user_password='"+ password +"';";
```

```
SELECT * FROM users WHERE user_name='user' AND user_password=''
OR 'a'='a';
```

INFORMATIKA UNPAR

Intermezzo: SQL injection (2)

- Example 2:

```
int expected_data = 1;
String sql = "SELECT * FROM users where id=" + expected_data;
```

```
String spoiled_data = "1; DROP TABLE users;"
String sql = "SELECT * FROM users where id=" + spoiled_data;
```

Prepared Statement

- To prevent SQL Injection, we can use prepared statement.

```
72      public Optional<Person> findById(int id) {
73          Connection connection = null;
74          PreparedStatement statement = null;
75          ResultSet resultSet = null;
76          try {
77              connection = dataSource.getConnection();
78              statement = connection.prepareStatement(
79                  "select id, fname, lname from Person where id=?");
80              statement.setInt(1, id);
81              resultSet = statement.executeQuery();
```

*the rest are omitted (same as before)

JDBC Template

- The code in basic JDBC feel little bit messy because we have to manually set each items needed.
- JDBC Template eliminates much of the boilerplate code required in basic JDBC:
  - Automatically use DataSource to open connections.
  - .query() use prepared statement.
  - Have RowMapper.
  - Automatically close connection after used.

JDBC Template - Implementation

```
29      @Autowired
30      private JdbcTemplate jdbcTemplate;
31
32      public Optional<Person2> findById(int id) {
33          List<Person2> results = jdbcTemplate.query(
34              "select id, fname, lname from Person where id=?",
35              this::mapRowToPerson,
36              id);
37          return results.size() == 0 ? Optional.empty() : Optional.of(results.get(0));
38      }
39
40      private Person2 mapRowToPerson(ResultSet resultSet, int rowNum) throws SQLException {
41          return new Person2(
42              resultSet.getInt("id"),
43              resultSet.getString("fname"),
44              resultSet.getString("lname")
45          );
46      }
```

Repository Pattern

- As explained before, when using repository pattern, we move the implementation of accessing database to class called Repository.
- First, we define the Repository interface:

```
5   public interface Person3Repository {
6       Iterable<Person3> findAll();
7       Optional<Person3> findById(int id);
8       Person3 save(Person3 data);
9   }
```

- Then, we implement the interface in JdbcPersonRepository using jdbc template.

Repository Pattern (2)

- In Controller, we are free from code for accessing database.

```
11  @Controller
12  @RequestMapping("/person3")
13  public class Person3Controller {
14
15      @Autowired
16      private Person3Repository repo;
17
18      @GetMapping("/")
19      public String index(Model model) {
20          Optional<Person3> person = this.repo.findById(1);
21          if(!person.isEmpty()){
22              model.addAttribute("name", person.get().getFullName());
23          }
24          return "index";
25      }
26  }
```

Extra: Spring Data

INFORMATIKA
UNPAR

Spring Data

- Definition: Spring Data is a part of the Spring Framework that simplifies database operations and provides consistent abstractions for working with different data sources (relational, NoSQL, etc.).
- Goal: Reduce the amount of boilerplate code needed to interact with various databases.
- Key Benefits:
  - Simplifies persistence layer.
  - Enables flexible query methods.
  - Supports various data stores like SQL, NoSQL, etc.
- Common: Spring Data JDBC / Spring Data JPA

Key Features of Spring Data

- **Repositories**: Provides abstraction layers for data access logic.
- **CrudRepository** and **JpaRepository**: Pre-built interfaces for common database operations.
- **Query Methods**: Automatically generate queries from method names.
- **Custom Queries**: Support for JPQL (Java Persistence Query Language) and native SQL queries.
- **NoSQL support**: Out-of-the-box support for MongoDB, Redis, Cassandra, etc.

Disadvantages:

- It's not always recommended for beginners to dive straight into using Spring Data:
  - Lack of Understanding of Core Concepts
    - Over-Abstraction
  - Hiding Too Much Complexity
    - Difficult when dealing complex query or optimization
  - Learning Curve with Spring Data Specifics
  - Limited Control Over Query Generation
    - Sometimes generate inefficient queries.
    - Difficult to modify.

Thanks

INFORMATIKA
UNPAR

INFORMATIKA
UNPAR

informatika.unpar.ac.id
informatika@unpar.ac.id
if.unpar
if.unpar