Spring JdbcTemplate In Clause

Raúl Estrada

Octubre 2020

1. Introduction

In a SQL statement, we can use the IN operator to test whether an expression matches any value in a list. Therefore, we can use the IN operator instead of multiple OR conditions.

In this tutorial, we'll show how to pass a list of values into the IN clause of a Spring JDBC template query.

2. Passing a *List* Parameter to *IN* Clause

The IN operator allows us to specify multiple values in a WHERE clause. For example, we can use it to find all employees whose id is in a specified id list:

```
SELECT * FROM EMPLOYEE WHERE id IN (1, 2, 3)
```

Typically, the total number of values inside the IN clause is variable. Therefore, we need to create a placeholder that can support a dynamic list of values.

2.1. With *JdbcTemplate*

With *JdbcTemplate*, we can use '?' characters as placeholders for the list of values. The number of '?' characters will be the same as the size of the list:

```
List<Employee> getEmployeesFromIdList(List<Integer> ids) {
    String inSql = String.join(",", Collections.nCopies(ids.size(), "?"));

    List<Employee> employees = jdbcTemplate.query(
      String.format("SELECT * FROM EMPLOYEE WHERE id IN (%s)", inSql),
      ids.toArray(),
      (rs, rowNum) -> new Employee(rs.getInt("id"), rs.getString("first_name"),
        rs.getString("last_name")));

    return employees;
}
```

In this method, we first generate a placeholder string that contains *ids.size()* '?' characters separated with commas. Then, we put this string into the IN clause of our SQL statement. For example, if we have three numbers in the *ids* list, the SQL statement is:

```
SELECT * FROM EMPLOYEE WHERE id IN (?,?,?)
```

In the *query* method, we pass the *ids* list as a parameter to match the placeholders inside the IN clause. This way, we can execute a dynamic SQL statement based on the input list of values.

2.2. With *NamedParameterJdbcTemplate*

Another way to handle the dynamic list of values is to use *NamedParameterJdbcTemplate*. For example, we can directly create a named parameter for the input list:

```
List<Employee> getEmployeesFromIdListNamed(List<Integer> ids) {
    SqlParameterSource parameters = new MapSqlParameterSource("ids", ids);

    List<Employee> employees = namedJdbcTemplate.query(
      "SELECT * FROM EMPLOYEE WHERE id IN (:ids)",
      parameters,
      (rs, rowNum) -> new Employee(rs.getInt("id"), rs.getString("first_name"),
        rs.getString("last_name")));

    return employees;
}
```

In this method, we first construct a *MapSqlParameterSource* object that contains the input id list. Then, we only use one named parameter to represent the dynamic list of values.

Under the hood, *NamedParameterJdbcTemplate* substitutes the named parameters for the '?' placeholders and uses *JdbcTemplate* to execute the query.

3. Handling a Large *List*

When we have a large number of values in a list, we should consider alternative ways to pass them into the *JdbcTemplate* query.

For example, the Oracle database doesn't support more than 1,000 literals in an IN clause.

One way to do that is to **create a temporary table for the list**. However, different databases can have different ways to create temporary tables. For example, we can use the *CREATE GLOBAL TEMPORARY TABLE* statement to create a temporary table in the Oracle database.

Let's create a temporary table for the H2 database:

```
List<Employee> getEmployeesFromLargeIdList(List<Integer> ids) {
    jdbcTemplate.execute("CREATE TEMPORARY TABLE IF NOT EXISTS employee_tmp (id INT NOT NULL)");

    List<Object[]> employeeIds = new ArrayList<>();
    for (Integer id : ids) {
        employeeIds.add(new Object[] { id });
    }
    jdbcTemplate.batchUpdate("INSERT INTO employee_tmp VALUES(?)", employeeIds);

    List<Employee> employees = jdbcTemplate.query(
      "SELECT * FROM EMPLOYEE WHERE id IN (SELECT id FROM employee_tmp)",
      (rs, rowNum) -> new Employee(rs.getInt("id"), rs.getString("first_name"),
      rs.getString("last_name")));

    jdbcTemplate.update("DELETE FROM employee_tmp");

    return employees;
}
```

Here, we first create a temporary table to hold all values of the input list. Then, we insert the input list's values into this table.

In our resulting SQL statement, **the values in the IN clause are from the temporary table**, and we've avoided constructing an IN clause with a large number of placeholders.

Finally, after we finish the query, we clean up the temporary table for future reuse.

4. Conclusion

In this tutorial, we showed how to use *JdbcTemplate* and *NamedParameterJdbcTemplate* to pass a list of values for the IN clause of a SQL query. Also, we provided an alternative way to handle a large number of list values by using a temporary table.