Spring - Difference Between RowMapper and ResultSetExtractor

Last Updated : 23 Jul, 2025

Understanding the difference between RowMapper and ResultSetExtractor is very important for anyone working with JDBC in Java. Both play important roles in fetching and processing data from the database. The main difference between RowMapper and ResultSetExtractor lies in their responsibilities.

- **RowMapper:** It is used to map each row of data from the resultSet to a Java object.
- **ResultsetExtractor:** It is the object that stores and allows navigation through the data retrieved from the database.

RowMapper vs ResultSetExtractor

The below table demonstrates the difference between RowMapper and ResultSetExtractor

Feature	RowMapper	ResultSetExtractor
Scope	Maps a single row at a time	Processes the entire ResultSet
Use Case	Simple row-to-object mapping	Complex ResultSet processing

Feature	RowMapper	ResultSetExtractor
Flexibiltiy	Less flexible, simpler to use	More flexible, allows custom logic
Method	mapRow(ResultSet rs, int rowNum)	extractData(ResultSet rs)
Common Usage	Used with JdbcTemplate.query()	Used with JdbcTemplate.query()

RowMapper

The RowMapper interface is designed to map a single row of a ResultSet to a Java object. It is used when the query returns multiple rows, and each row needs to be mapped to a corresponding object.

- **Single Row Mapping:** Maps one row at a time using the mapRow()method.
- **Simplified Code:** Converts rows into objects, improving readability and maintainability.
- **Common Use Case:** Often used with JdbcTemplate.query() for automatic mapping of ResultSet rows.

Example: This example demonstrates how to use a custom RowMapper (EmployeeRowMapper) with JdbcTemplate to map rows from a ResultSet to Employee objects.

```
public class EmployeeRowMapper implements RowMapper<Employee> {
    @Override
    public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {
        Employee employee = new Employee();
        employee.setId(rs.getInt("id"));
        employee.setName(rs.getString("name"));
        employee.setSalary(rs.getDouble("salary"));
        return employee;
    }
```

```
}

// Usage with JdbcTemplate
List<Employee> employees = jdbcTemplate.query("SELECT * FROM employee", new
EmployeeRowMapper());
```

ResultSetExtractor

The ResultSetExtractor interface is more flexible and powerful. It is designed to process the entire ResultSet at once, allowing for more complex mappings.

- **Entire ResultSet Processing:** Processes the whole ResultSet for complex operations or aggregations.
- **Custom Logic:** Allows implementation of custom logic for advanced use cases.
- **Common Use Case:** Used with JdbcTemplate.query() for custom processing beyond simple row mapping.

Example: This example demonstrates how to use a custom ResultSetExtractor (EmployeeResultSetExtractor) with JdbcTemplate to process the entire ResultSet and map multiple rows to a list of Employee objects.

```
public class EmployeeResultSetExtractor implements
ResultSetExtractor<List<Employee>> {
    @Override
    public List<Employee> extractData(ResultSet rs) throws SQLException,
DataAccessException {
        List<Employee> employees = new ArrayList<>();
        while (rs.next()) {
            Employee employee = new Employee();
            employee.setId(rs.getInt("id"));
            employee.setName(rs.getString("name"));
            employee.setSalary(rs.getDouble("salary"));
            employees.add(employee);
        }
        return employees;
    }
}

// Usage with JdbcTemplate
```

```
List<Employee> employees = jdbcTemplate.query("SELECT * FROM employee", new
EmployeeResultSetExtractor());
```

Note:

- Use RowMapper for straightforward row-to-object mapping, ideal for simple queries.
- Use ResultSetExtractor for complex processing, like aggregating data or mapping multiple rows to a single object.