

# Spring Boot + Apache Camel SQL component + Transaction Management Example

---

 [javainuse.com/camel/camel\\_bootstrapsqltransact](http://javainuse.com/camel/camel_bootstrapsqltransact)

```

package com.javainuse.service.impl;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.sql.DataSource;

import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.sql.SqlComponent;
import org.apache.camel.spring.spi.SpringTransactionPolicy;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.stereotype.Service;
import org.springframework.transaction.PlatformTransactionManager;

import com.javainuse.model.Employee;

@Service
public class EmployeeServiceImpl extends RouteBuilder {

    @Autowired
    DataSource dataSource;

    public DataSource getDataSource() {
        return dataSource;
    }

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    @Bean
    public SqlComponent sql(DataSource dataSource) {
        SqlComponent sql = new SqlComponent();
        sql.setDataSource(dataSource);
        return sql;
    }

    // Create Transaction Manager
    @Bean
    public PlatformTransactionManager transactionManager(DataSource
dataSource) {
        return new DataSourceTransactionManager(dataSource);
    }

    // Specify Spring Transaction Management Policy
    @Bean(name = "PROPAGATION_REQUIRED")
    public SpringTransactionPolicy
propagationRequired(PlatformTransactionManager transactionManager) {
        SpringTransactionPolicy propagationRequired = new
SpringTransactionPolicy();
        propagationRequired.setTransactionManager(transactionManager);
    }
}

```

```

propagationRequired.setPropagationBehaviorName("PROPAGATION_REQUIRED");
    return propagationRequired;
}

@Override
public void configure() throws Exception {

    //Insert Route

    from("direct:insert").transacted("PROPAGATION_REQUIRED").log("Processing message:
").process(new Processor() {
        public void process(Exchange xchg) throws Exception {
            //take the Employee object from the exchange and
            create the parameter map
            Employee employee =
            xchg.getIn().getBody(Employee.class);
            Map<String, Object> employeeMap = new
            HashMap<String, Object>();
            employeeMap.put("EmpId", employee.getEmpId());
            employeeMap.put("EmpName",
            employee.getEmpName());
            xchg.getIn().setBody(employeeMap);
        }
    }).to("sql:INSERT INTO employee(EmpId, EmpName) VALUES (:#EmpId,
    :#EmpName)").

        process(new Processor() {
            public void process(Exchange xchg) throws
            Exception {
                // throw an exception after insert
                System.out.println("Exception
                Occurred");
                throw new Exception();
            }
        });

    // Select Route
    from("direct:select").to("sql:select * from employee").process(new
    Processor() {
        public void process(Exchange xchg) throws Exception {
            //the camel sql select query has been executed. We get
            the list of employees.
            ArrayList<Map<String, String>> dataList =
            (ArrayList<Map<String, String>>) xchg.getIn().getBody();
            List<Employee> employees = new
            ArrayList<Employee>();
            System.out.println(dataList);
            for (Map<String, String> data : dataList) {
                Employee employee = new Employee();
                employee.setEmpId(data.get("empId"));
                employee.setEmpName(data.get("empName"));
                employees.add(employee);
            }
            xchg.getIn().setBody(employees);
        }
    });
}

```

```
}  
}
```

Restart the application again. Again send the POST request. An exception is thrown, but this time the insertions get reverted.

Method: POST, Request URL: http://localhost:8080/employees, SEND button, and a menu icon.

Parameters: Headers, Body, Variables

Header name: content-type, Header value: application/json, X icon, and a pencil icon.

ADD HEADER

Headers are valid, Headers size: 30 bytes

500 Internal Server Error, 891.11 ms, DETAILS

```
{  
  "timestamp": "2020-03-27T17:58:52.911+0000",  
  "status": 500,  
  "error": "Internal Server Error",  
  "message": "Exception occurred during execution on the exchange: Exchange[ID-LAPTOP-AFRDD9N9-55673-1585331904103-0-2]",  
  "path": "/employees"  
}
```

If we check the database there will not be any records

Download Source Code

```
mysql> select * from employee;  
Empty set (0.04 sec)
```

Download it -

Spring Boot + Apache Camel SQL component + Transaction Management

[See Also](#)

[Spring Boot Hello World Application- Create simple controller and jsp view using Maven](#)

[Spring Boot Tutorial-Spring Data JPA](#)

[Spring Boot + Simple Security Configuration](#)

[Pagination using Spring Boot Simple Example](#)

[Spring Boot + ActiveMQ Hello world Example](#)

[Spring Boot + Swagger Example Hello World Example](#)

[Spring Boot + Swagger- Understanding the various Swagger Annotations](#)

[Spring Boot Main Menu](#)

[Spring Boot Interview Questions](#)

[Limited time offer: Get 10 free Adobe Stock images.ads via Carbon](#)

In [previous tutorial](#) we had implemented [Spring Boot + Apache Camel SQL Component + MySQL Example](#) for inserting and retrieving records from MySQL. **For this tutorial we had not implemented any Transaction Management, so all transactions were auto commit by default.** In this tutorial we

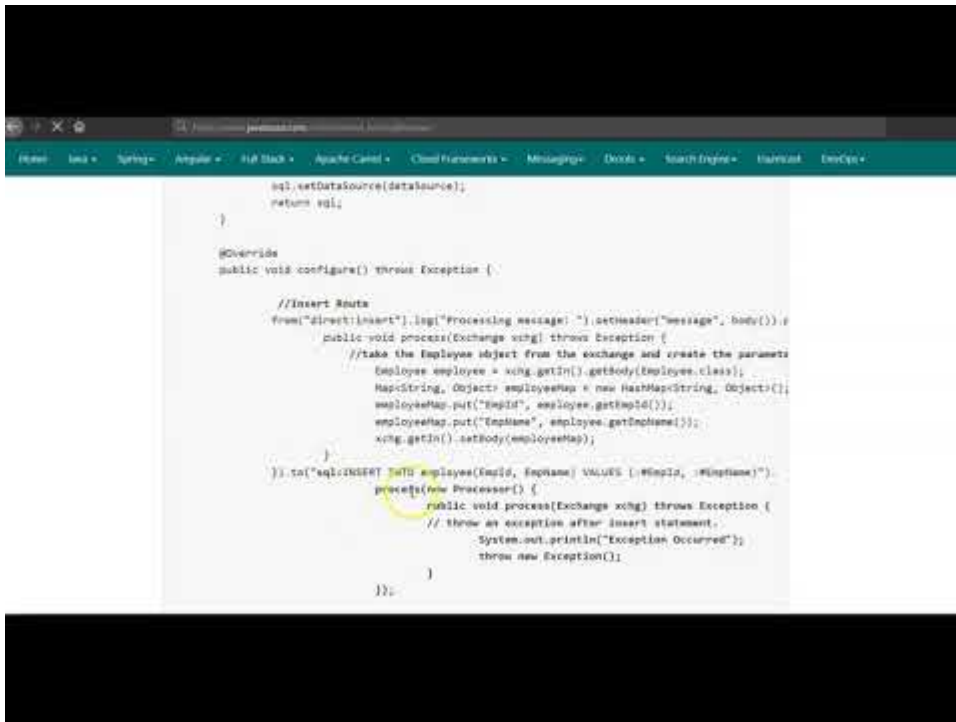
will be implementing transaction Management for the Apache Camel Insert Route we had implemented before. As we are making use of Apache Camel SQL component which in turn makes use of Spring JDBC we will be implementing transaction management using Spring Transaction. In [previous tutorial series](#) we had implemented [Spring Boot Transaction Management Examples](#). We had also taken a look at [what is Transaction Propagation and how to implement it using Spring Boot](#). In this tutorial we will be making use of transaction propagation type of **PROPAGATION\_REQUIRED**.

We will be [modifying the code we had developed in previous tutorial for Apache Camel + Spring SQL Component + MySQL Example](#)

[Video](#)

This tutorial is explained in the below Youtube Video.





Watch Video At: <https://youtu.be/mPLJxQVlYg4>

## Apache Camel - Table of Contents

File Transfer Using Java DSL Apache Camel

Apache Camel Java DSL + Spring Integration Hello World Example

Apache Camel Exception Handling Using Simple Example

Apache Camel Redelivery policy using example

Integrate Apache Camel and ActiveMQ

EIP patterns using Apache Camel

Apache Camel Tutorial- Integrate Spring Boot+ Apache Camel

Apache Camel Tutorial- Integrate with MySQL DB using SQL query.

Apache Camel EIP - Splitter and Aggregator pattern

Apache Camel Unit Testing

Apache Camel + Spring + Quartz Hello World Example

Camel application deployment on JBoss Fuse

Apache Camel + Apache CXF SOAP Webservices

Apache Camel + JAX-RS REST Webservice

Apache Camel + CXFRS REST Webservice

Apache Camel Routing Slip EIP Pattern

Apache Camel Dynamic Router Pattern

Apache Camel Load Balancer EIP Pattern

Apache Camel Interceptors

Apache Camel + Kafka Hello World Example

Apache Camel - Marshalling/Unmarshalling XML/JSON Data Example

Calling and Consuming Webservices using Apache Camel

Apache Camel Tutorial - Send SMTP Email Using Gmail

Apache Camel Tutorial - SEDA component Hello World Example

Apache Camel Tutorial - Idempotent Consumer using MemoryIdempotentRepository and FileIdempotentRepository

Spring Boot + Apache Camel + RabbitMQ - Hello World Example

Spring Boot + Apache Camel JDBC component + MySQL - Hello World Example

Spring Boot + Apache Camel SQL component + MySQL - Hello World Example

Spring Boot + Apache Camel SQL component + Transaction Management Example

## **Exception thrown in Camel route without Transaction Management**

---

Initially we will be taking a look at the application behaviour without transaction Management. So we will be throwing an exception after the values are inserted in MySQL database. As no transaction management has been implemented so auto commit is true and the insert is not rolled back. We will modify the Service class where we have configured the camel route. In the insert route after inserting the data we will be throwing an exception.

```

package com.javainuse.service.impl;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.sql.DataSource;

import org.apache.camel.Exchange;
import org.apache.camel.Processor;
import org.apache.camel.builder.RouteBuilder;
import org.apache.camel.component.sql.SqlComponent;
import org.apache.camel.spring.spi.SpringTransactionPolicy;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;
import org.springframework.stereotype.Service;
import org.springframework.transaction.PlatformTransactionManager;

import com.javainuse.model.Employee;

@Service
public class EmployeeServiceImpl extends RouteBuilder {

    @Autowired
    DataSource dataSource;

    public DataSource getDataSource() {
        return dataSource;
    }

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

    @Bean
    public SqlComponent sql(DataSource dataSource) {
        SqlComponent sql = new SqlComponent();
        sql.setDataSource(dataSource);
        return sql;
    }

    @Override
    public void configure() throws Exception {

        //Insert Route
        from("direct:insert").log("Processing message:
").setHeader("message", body()).process(new Processor() {
            public void process(Exchange xchg) throws Exception {
                //take the Employee object from the exchange and
                create the parameter map
                Employee employee =
xchg.getIn().getBody(Employee.class);
                Map<String, Object> employeeMap = new
HashMap<String, Object>();
                employeeMap.put("EmpId", employee.getEmpId());
                employeeMap.put("EmpName", employee.getEmpName());
                xchg.getIn().setBody(employeeMap);
            }
        });
    }
}

```



```

        }
    }).to("sql:INSERT INTO employee(EmpId, EmpName) VALUES (:#EmpId,
:#EmpName)").

        process(new Processor() {
            public void process(Exchange xchg) throws
Exception {
                // throw an exception after insert

                System.out.println("Exception
Occurred");

                throw new Exception();
            }
        });

    // Select Route
    from("direct:select").to("sql:select * from employee").process(new
Processor() {
        public void process(Exchange xchg) throws Exception {
            //the camel sql select query has been executed. We get the
list of employees.

            ArrayList<Map<String, String>> dataList =
(ArrayList<Map<String, String>>) xchg.getIn().getBody();
            List<Employee> employees = new ArrayList<Employee>
();

            System.out.println(dataList);
            for (Map<String, String> data : dataList) {
                Employee employee = new Employee();
                employee.setEmpId(data.get("empId"));
                employee.setEmpName(data.get("empName"));
                employees.add(employee);
            }
            xchg.getIn().setBody(employees);
        }
    });
}
}
}

```

Start the application.

Send a POST request to insert new employee detail-

Method

Request URL

POST

http://localhost:8080/employees

SEND

Parameters ^

Headers

Body

Variables

<> Toggle source mode

+ Insert headers set

Header name

Header value

content-type

application/json

ADD HEADER

✓ Headers are valid

Headers size: 30 bytes

500 Internal Server Error

891.11 ms

DETAILS v

<>

```

{
  "timestamp": "2020-03-27T17:58:52.911+0000",
  "status": 500,
  "error": "Internal Server Error",
  "message": "Exception occurred during execution on the exchange: Exchange[ID-LAPTOP-AFRDD9N9-55673-1585331904103-0-2]",
  "path": "/employees"
}

```

This will throw an exception. However we observe that the insertion in employee table is not reverted. So it is auto commit by default.

```
mysql> select * from employee;
+-----+-----+
| empId | empName |
+-----+-----+
| emp001 | emp1    |
+-----+-----+
1 row in set (0.00 sec)
```

## Exception thrown in Camel route with Transaction Management

We now define transaction management for the Apache Camel Route. We do so making use of the Camel SpringTransactionPolicy. So after the exception is thrown the insert statement will be rolled back.