* [Home](https://www.javainuse.com/home)
* [Java](https://www.javainuse.com/)
* [Spring](https://www.javainuse.com/)
* [Angular](https://www.javainuse.com/)
* [Full Stack](https://www.javainuse.com/)
* [Apache Camel](https://www.javainuse.com/)
* [Cloud Frameworks](https://www.javainuse.com/)
* [Messaging](https://www.javainuse.com/)
* [Drools](https://www.javainuse.com/)
* [Search Engine](https://www.javainuse.com/)
* [Hazelcast](https://www.javainuse.com/hazelcast)
* [DevOps](https://www.javainuse.com/)

Search Tutorials

Top of Form

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  | | --- | --- | |  |  | |  |

Bottom of Form

Spring Boot Declarative Transaction Management Example

In previous tutorial - [Spring Boot + JDBC Example](https://www.javainuse.com/spring/bootjdbc) we implemented JDBC using Spring boot with MySql database. In this tutorial we will be understanding what is transaction management and implement them for an application developed using Spring Boot + JDBC. In the next tutorial we will be implementing [Transaction Propagation using Spring Boot.](https://www.javainuse.com/spring/boot-transaction-propagation)

Spring Boot Transaction Management - Table of Contents

[**Spring Boot Transaction Management Example**](https://www.javainuse.com/spring/boot-transaction)

[Spring Boot Transactions - Understanding Transaction Isolation](https://www.javainuse.com/spring/boot-transaction-isolation)

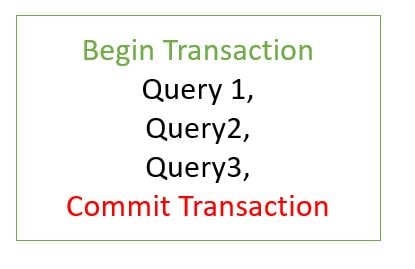
[Spring Boot Transactions - Understanding Transaction Propagation](https://www.javainuse.com/spring/boot-transaction-propagation)

[Spring Boot Transactions - Understanding Transaction Rollbacks](https://www.javainuse.com/spring/boot-rollback)

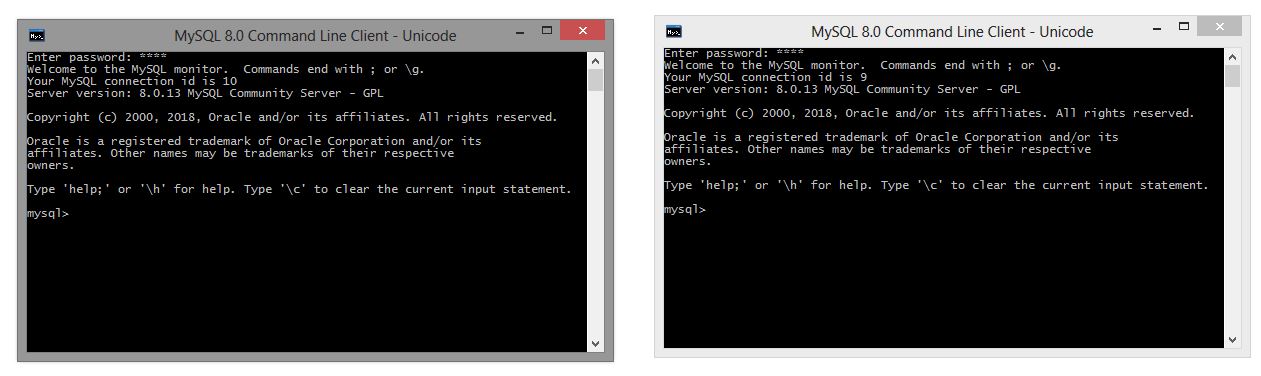
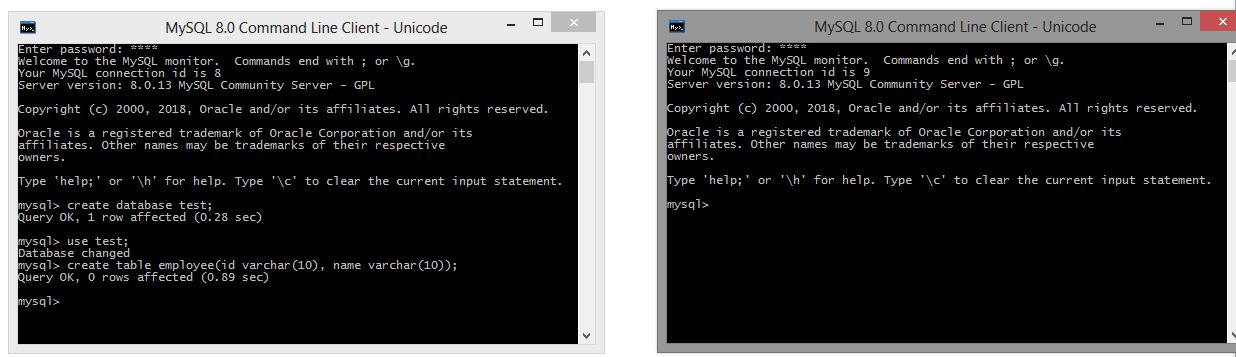
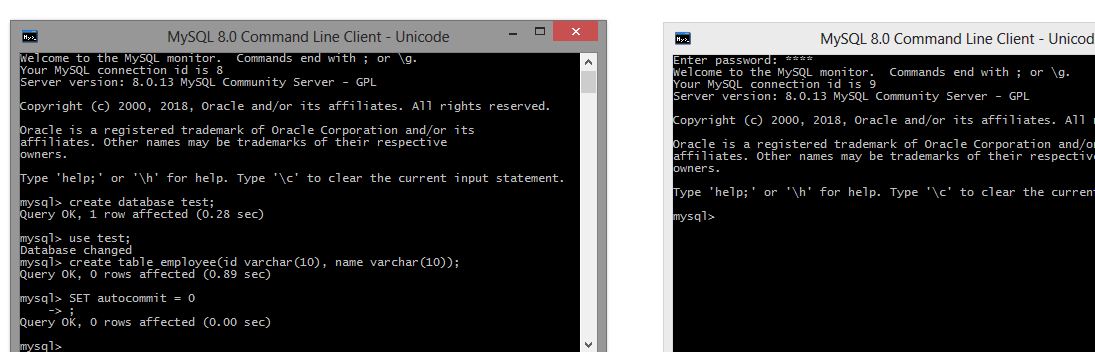
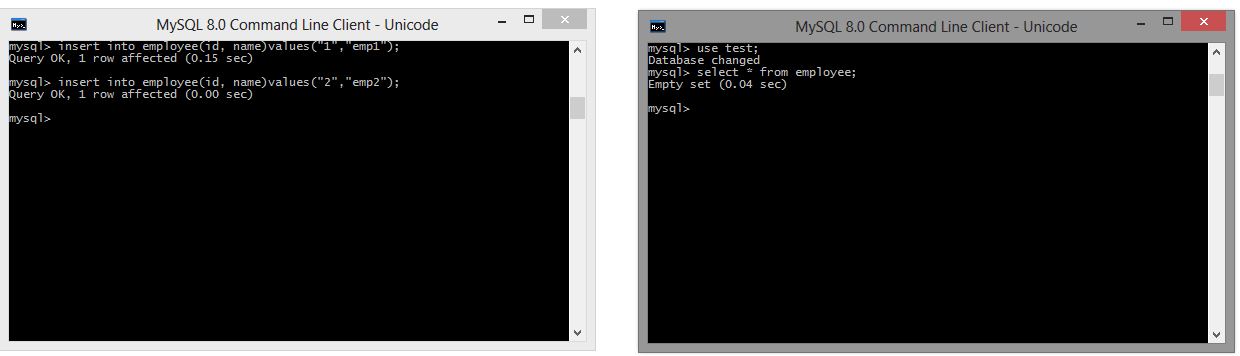
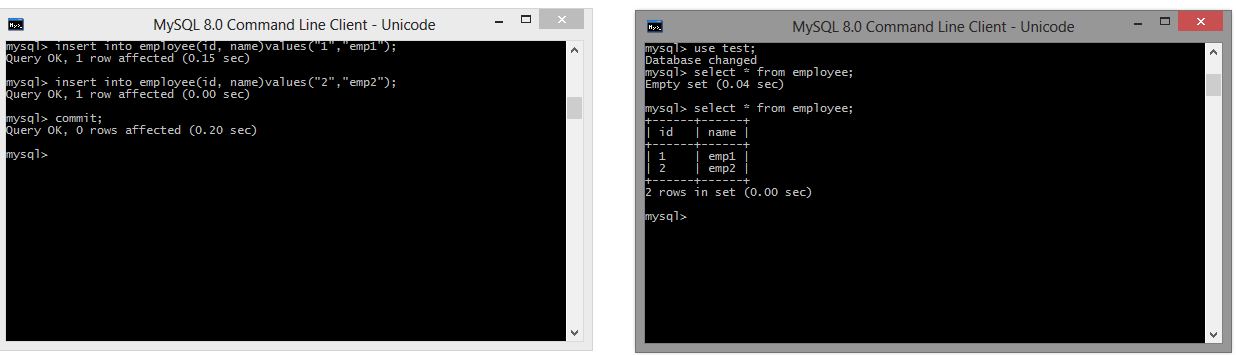
Video

This tutorial is explained in the below Youtube Video.

Lets Begin-

What are Database Transactions?  
A Database transaction is a **single logical unit of work which accesses and possibly modifies the contents of a database.**  
  


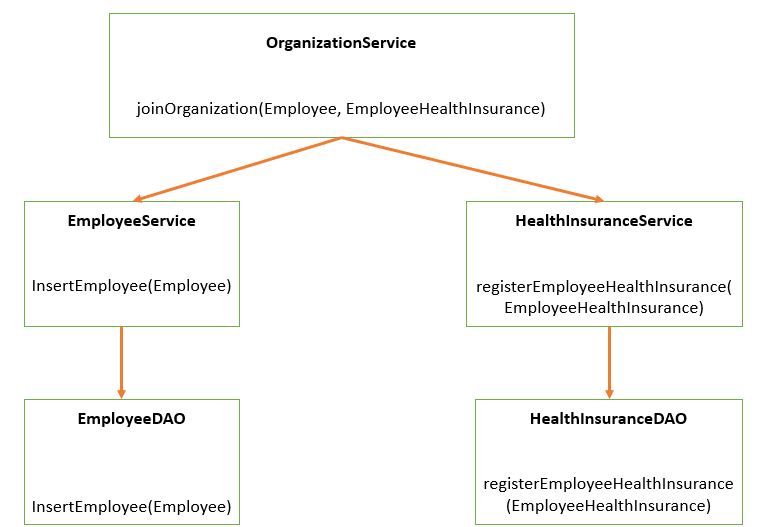
Let us check this for the mysql database.

* Open two separate windows for the mysql database.  
  
* In one mysql window create a database named test and in it a table named employee  
  
* By default the transactions are **autocommit** for mysql database.  
  We will disable autocommit using following command- SET autocommit = 0  
  
* In first mysql window use the following insert commands- If we now using the second mysql window do a select for the employee table we will not see any records. This is because the **transactions are still not committed in the first mysql window.**  
  
* We now use the **commit command** in the first MySQL command. If we now using the second mysql window do a select for the employee table we will see the two records.  
  

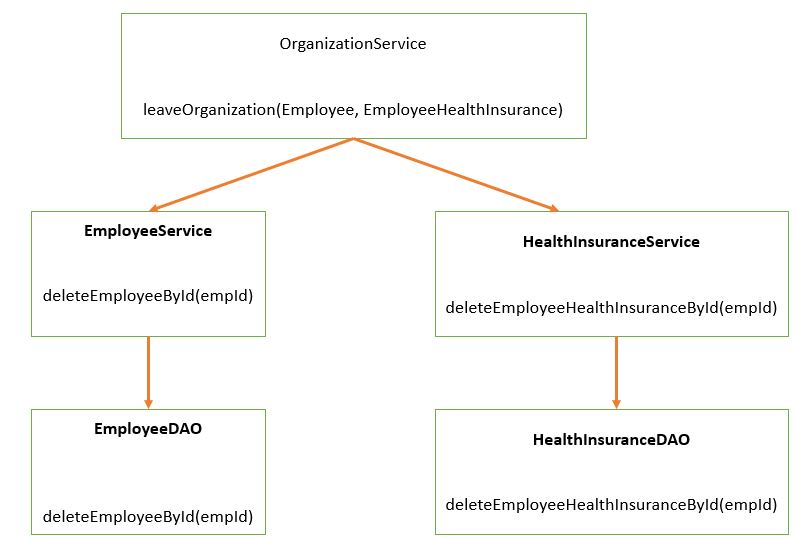
Let us now use application transaction for **Spring Boot JDBC project.**  
We will be developing a Spring Boot + JDBC project for employee management. It will be having 3 services-

* **EmployeeService -** The service will perform Employee Operations
* **HealthInsuranceService -** The service will perform Employee Health Insurance Operations
* **OrganizationService -** The service will perform Organization Level Operation like Employee joining and exit. It makes use of the EmployeeService and HealthInsuranceService

OrganizationService Employee Join Workflow

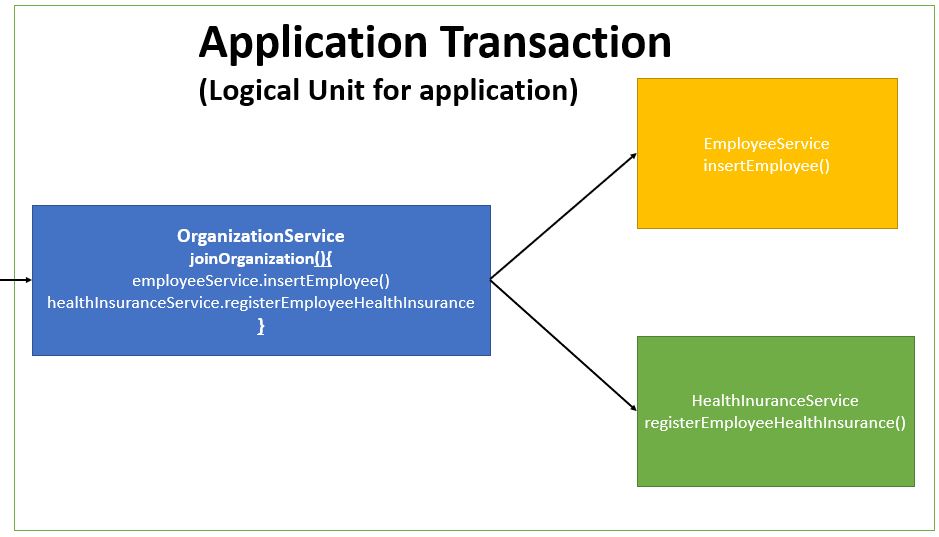
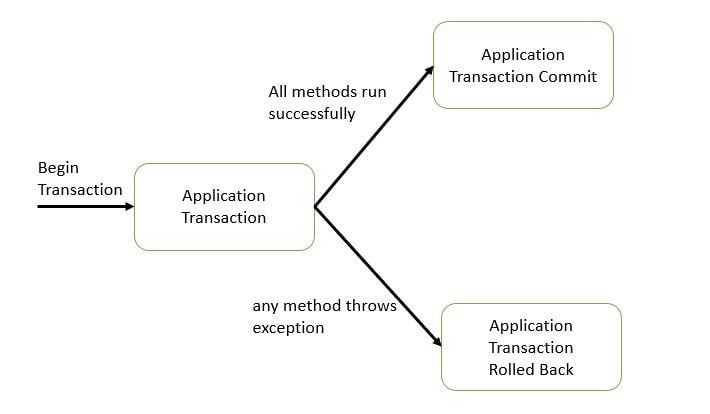
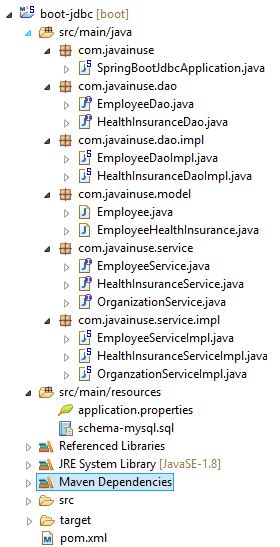


OrganizationService Employee Exit Workflow



An **application transaction is a sequence of application actions that are considered as a single logical unit** by the application. For our application the joinOrganization method will be considered as one complete transaction. joinOrganization consists of two actions-

* Persist Employee Information
* Persist HealthInsurance Information

  
If due to any reason any one of the above action fails then the other action should also be roll backed. So if Employee Information gets inserted but suppose due to some reason persist HealthInsurance is not successful, then Employee Information should also be rollbacked. **It means it is all or none for a logical unit of work.** Similar will be the case for exitOrganization Method which will be considered as one unit of work.  
  
Initially we will not be using any transaction management. By default the **spring boot transaction is auto commit.** But this is not a good practice we will see why in the next section.  
The maven project will be as follows-  
  
The pom.xml will be as follows-

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.javainuse</groupId>

<artifactId>boot-jdbc</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>boot-jdbc</name>

<description>Demo project for Spring Boot</description>

<parent>

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

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

<version>2.1.1.RELEASE</version>

<relativePath /> <!-- lookup parent from repository -->

</parent>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

**<dependency>**

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

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

**</dependency>**

<dependency>

<groupId>mysql</groupId>

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

<scope>runtime</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

Create the application.properties as follows -

spring.datasource.url=jdbc:mysql://localhost/bootdb?createDatabaseIfNotExist=true&autoReconnect=true&useSSL=false

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.platform=mysql

spring.datasource.initialization-mode=always

logging.level.org.springframework=DEBUG

Create the schema-mysql.sql as follows. This is the initialization script which is run at the beginning by Spring Boot JDBC-

DROP TABLE IF EXISTS employee;

DROP TABLE IF EXISTS employeeHealthInsurance;

CREATE TABLE employee (

empId VARCHAR(10) NOT NULL,

empName VARCHAR(100) NOT NULL

);

CREATE TABLE employeeHealthInsurance (

empId VARCHAR(10) NOT NULL,

healthInsuranceSchemeName VARCHAR(100) NOT NULL,

coverageAmount VARCHAR(100) NOT NULL

);

Define the Model class Employee which will represent the Employee details-

package com.javainuse.model;

public class Employee {

private String empId;

private String empName;

public String getEmpId() {

return empId;

}

public void setEmpId(String empId) {

this.empId = empId;

}

public String getEmpName() {

return empName;

}

public void setEmpName(String empName) {

this.empName = empName;

}

@Override

public String toString() {

return "Employee [empId=" + empId + ", empName=" + empName + "]";

}

}

Create the EmployeeDAO interface for performing Employee operations as follows-

package com.javainuse.dao;

import com.javainuse.model.Employee;

public interface EmployeeDao {

void insertEmployee(Employee cus);

void deleteEmployeeById(String empid);

}

Create the EmployeeDAOImpl which implements the EmployeeDAO interface as follows. Spring Boot will detect spring-jdbc on the classpath and mysql and will create a DataSource and a JdbcTemplate for us automatically. Because such infrastructure is now available and we have no dedicated configuration, a DataSourceTransactionManager will also be created for us.

package com.javainuse.dao.impl;

import javax.annotation.PostConstruct;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.support.JdbcDaoSupport;

import org.springframework.stereotype.Repository;

import com.javainuse.dao.EmployeeDao;

import com.javainuse.model.Employee;

@Repository

public class EmployeeDaoImpl extends JdbcDaoSupport implements EmployeeDao {

@Autowired

DataSource dataSource;

@PostConstruct

private void initialize() {

setDataSource(dataSource);

}

@Override

public void insertEmployee(Employee emp) {

String sql = "INSERT INTO employee " + "(empId, empName) VALUES (?, ?)";

getJdbcTemplate().update(sql, new Object[] { emp.getEmpId(), emp.getEmpName() });

}

@Override

public void deleteEmployeeById(String empid) {

String sql = "DELETE FROM employee WHERE empId = ?";

getJdbcTemplate().update(sql, new Object[] { empid });

}

}

Define the Model class EmployeeHealthInsurance which will represent the Employee Health Insurance details-

package com.javainuse.model;

public class EmployeeHealthInsurance {

private String empId;

private String healthInsuranceSchemeName;

private int coverageAmount;

public String getEmpId() {

return empId;

}

public void setEmpId(String empId) {

this.empId = empId;

}

public String getHealthInsuranceSchemeName() {

return healthInsuranceSchemeName;

}

public void setHealthInsuranceSchemeName(String healthInsuranceSchemeName) {

this.healthInsuranceSchemeName = healthInsuranceSchemeName;

}

public int getCoverageAmount() {

return coverageAmount;

}

public void setCoverageAmount(int coverageAmount) {

this.coverageAmount = coverageAmount;

}

}

Create the HealthInsuranceDao for performing health insurance operations as follows-

package com.javainuse.dao;

import com.javainuse.model.EmployeeHealthInsurance;

public interface HealthInsuranceDao {

void registerEmployeeHealthInsurance(EmployeeHealthInsurance employeeHealthInsurance);

void deleteEmployeeHealthInsuranceById(String empid);

}

Create the EmployeeHealthInsuranceDAOImpl which implements HealthInsuranceDao as follows-

package com.javainuse.dao.impl;

import javax.annotation.PostConstruct;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.support.JdbcDaoSupport;

import org.springframework.stereotype.Repository;

import com.javainuse.dao.HealthInsuranceDao;

import com.javainuse.model.EmployeeHealthInsurance;

@Repository

public class HealthInsuranceDaoImpl extends JdbcDaoSupport implements HealthInsuranceDao {

@Autowired

DataSource dataSource;

@PostConstruct

private void initialize() {

setDataSource(dataSource);

}

@Override

public void registerEmployeeHealthInsurance(EmployeeHealthInsurance emp) {

String sql = "INSERT INTO employeeHealthInsurance "

+ "(empId, healthInsuranceSchemeName, coverageAmount) VALUES (?, ?,?)";

getJdbcTemplate().update(sql,

new Object[] { emp.getEmpId(), emp.getHealthInsuranceSchemeName(), emp.getCoverageAmount() });

}

@Override

public void deleteEmployeeHealthInsuranceById(String empid) {

String sql = "DELETE FROM employeeHealthInsurance WHERE empId = ?";

getJdbcTemplate().update(sql, new Object[] { empid });

}

}

Create the EmployeeService interface for performing employee operations as follows-

package com.javainuse.service;

import com.javainuse.model.Employee;

public interface EmployeeService {

void insertEmployee(Employee emp);

void deleteEmployeeById(String empid);

}

Create the EmployeeServiceImpl which implements the EmployeeService as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.javainuse.dao.EmployeeDao;

import com.javainuse.model.Employee;

import com.javainuse.service.EmployeeService;

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

EmployeeDao employeeDao;

@Override

public void insertEmployee(Employee employee) {

employeeDao.insertEmployee(employee);

}

@Override

public void deleteEmployeeById(String empid) {

employeeDao.deleteEmployeeById(empid);

}

}

Create the HealthInsuranceService interface as follows-

package com.javainuse.service;

import com.javainuse.model.EmployeeHealthInsurance;

public interface HealthInsuranceService {

void registerEmployeeHealthInsurance(EmployeeHealthInsurance employeeHealthInsurance);

void deleteEmployeeHealthInsuranceById(String empid);

}

Create the HealthInsuranceServiceImpl which implements the HealthInsuranceService as follows-

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.javainuse.dao.HealthInsuranceDao;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.HealthInsuranceService;

@Service

public class HealthInsuranceServiceImpl implements HealthInsuranceService {

@Autowired

HealthInsuranceDao healthInsuranceDao;

@Override

public void registerEmployeeHealthInsurance(EmployeeHealthInsurance employeeHealthInsurance) {

healthInsuranceDao.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

public void deleteEmployeeHealthInsuranceById(String empid) {

healthInsuranceDao.deleteEmployeeHealthInsuranceById(empid);

}

}

Create the OrganizationService interface as follows-

package com.javainuse.service;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

public interface OrganizationService {

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance);

public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance);

}

Create the OrganizationServiceImpl which implements the OrganizationService. It makes use of the EmployeeService and the HealthInsuranceService.

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.EmployeeService;

import com.javainuse.service.HealthInsuranceService;

import com.javainuse.service.OrganizationService;

@Service

public class OrganzationServiceImpl implements OrganizationService {

@Autowired

EmployeeService employeeService;

@Autowired

HealthInsuranceService healthInsuranceService;

@Override

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.insertEmployee(employee);

healthInsuranceService.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.deleteEmployeeById(employee.getEmpId());

healthInsuranceService.deleteEmployeeHealthInsuranceById(employeeHealthInsurance.getEmpId());

}

}

Finally create the Spring Boot Main class as follows-

package com.javainuse;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.OrganizationService;

@SpringBootApplication

public class SpringBootJdbcApplication {

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(SpringBootJdbcApplication.class, args);

OrganizationService organizationService = context.getBean(OrganizationService.class);

Employee emp = new Employee();

emp.setEmpId("emp1");

emp.setEmpName("emp1");

EmployeeHealthInsurance employeeHealthInsurance = new EmployeeHealthInsurance();

employeeHealthInsurance.setEmpId("emp1");

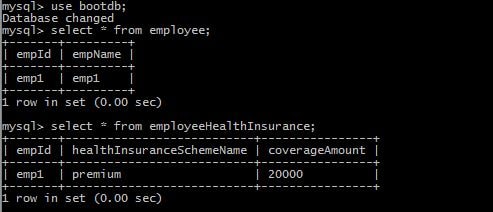
employeeHealthInsurance.setHealthInsuranceSchemeName("premium");

employeeHealthInsurance.setCoverageAmount(20000);

organizationService.joinOrganization(emp, employeeHealthInsurance);

}

}

If we now run the application, record will be inserted in both the employee table and the employeehealthinsurance table  
  
Suppose the employeeService call is successful but due to some reason the healthInsuranceService call fails. What should happen in this case. In such a scenario the entry made in the employee table for the new employee should also be reverted. Let us see how our application will behave in such a scenario.  
We are manually throwing an unchecked exception after the first service call is made.

package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.EmployeeService;

import com.javainuse.service.HealthInsuranceService;

import com.javainuse.service.OrganizationService;

@Service

public class OrganzationServiceImpl implements OrganizationService {

@Autowired

EmployeeService employeeService;

@Autowired

HealthInsuranceService healthInsuranceService;

@Override

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.insertEmployee(employee);

**if (employee.getEmpId().equals("emp1")) {**

**throw new RuntimeException("thowing exception to test transaction rollback");**

**}**

healthInsuranceService.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

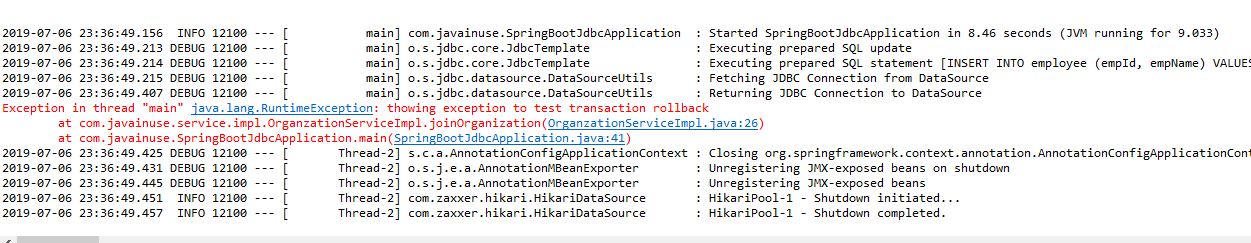
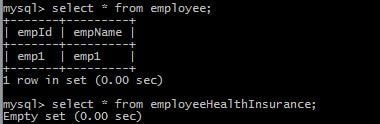
public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.deleteEmployeeById(employee.getEmpId());

healthInsuranceService.deleteEmployeeHealthInsuranceById(employeeHealthInsurance.getEmpId());

}

}

Lets now run the application.  
  
We can see that there is record in employee table but not in employeehealthinsurance table.  
  
Now let us implement transaction management. We will be using the **Transactional annotation.** Transaction is a cross cutting concern and it is implemented using AOP in Spring Boot.  


package com.javainuse.service.impl;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import com.javainuse.model.Employee;

import com.javainuse.model.EmployeeHealthInsurance;

import com.javainuse.service.EmployeeService;

import com.javainuse.service.HealthInsuranceService;

import com.javainuse.service.OrganizationService;

@Service

public class OrganzationServiceImpl implements OrganizationService {

@Autowired

EmployeeService employeeService;

@Autowired

HealthInsuranceService healthInsuranceService;

@Override

**@Transactional**

public void joinOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

employeeService.insertEmployee(employee);

if (employee.getEmpId().equals("emp1")) {

throw new RuntimeException("thowing exception to test transaction rollback");

}

healthInsuranceService.registerEmployeeHealthInsurance(employeeHealthInsurance);

}

@Override

**@Transactional**

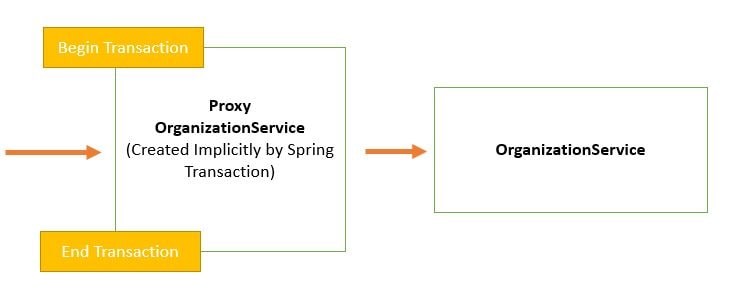
public void leaveOrganization(Employee employee, EmployeeHealthInsurance employeeHealthInsurance) {

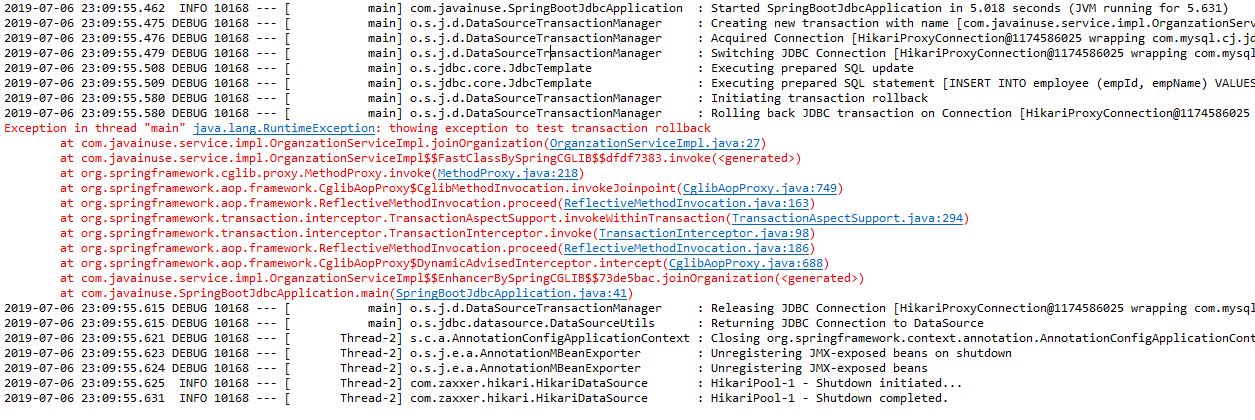
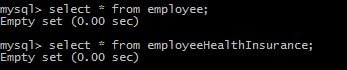
employeeService.deleteEmployeeById(employee.getEmpId());

healthInsuranceService.deleteEmployeeHealthInsuranceById(employeeHealthInsurance.getEmpId());

}

}

Spring Boot implicitly creates a **proxy for the transaction annotated methods.** So for such methods the proxy acts like a wrapper which takes care of creating a transaction at the beginning of the method call and committing the transaction after the method is executed.  


The component that intercepts the @Transactional annotated method like the EmployeeService. Now let us run the application again.  
  
If we now check the employee and the employeehealthinsurance table there are no records in both so our records are getting roll backed correctly.  


Download Source Code

Download it -  
[Spring Boot Transaction Management](https://www.javainuse.com/zip/spring/boot/boot-jdbc-transaction.rar)

Popular Posts

* [E-commerce Website - Online Book Store using Angular 8 + Spring Boot](https://www.javainuse.com/fullstack/ecommerce)
* [Spring Boot +JSON Web Token(JWT) Hello World Example](https://www.javainuse.com/spring/boot-jwt)
* [Angular 7 + Spring Boot Application Hello World Example](https://www.javainuse.com/spring/ang7-hello)
* [Build a Real Time Chat Application using Spring Boot + WebSocket + RabbitMQ](https://www.javainuse.com/spring/boot-websocket-chat)
* [Pivotal Cloud Foundry Tutorial - Deploy Spring Boot Application Hello World Example](https://www.javainuse.com/pcf/pcf-hello)
* [Deploying Spring Based WAR Application to Docker](https://www.javainuse.com/devOps/docker/docker-war)
* [EIP patterns using Apache Camel](https://www.javainuse.com/camel/camel_EIP)
* [Spring Cloud- Netflix Eureka + Ribbon Simple Example](https://www.javainuse.com/spring/spring_ribbon)
* [Spring Cloud- Netflix Hystrix Circuit Breaker Simple Example](https://www.javainuse.com/spring/spring_hystrix_circuitbreaker)
* [Spring Boot + Swagger Example Hello World Example](https://www.javainuse.com/spring/boot_swagger)
* [Spring Boot Batch Simple example](https://www.javainuse.com/spring/bootbatch)
* [Spring Boot + Apache Kafka Example](https://www.javainuse.com/spring/spring-boot-apache-kafka-hello-world)
* [Spring Boot Admin Simple Example](https://www.javainuse.com/spring/boot-admin)
* [Spring Boot Security - Introduction to OAuth](https://www.javainuse.com/spring/spring-boot-oauth-introduction)
* [Spring Boot OAuth2 Part 1 - Getting The Authorization Code](https://www.javainuse.com/spring/spring-boot-oauth-authorization-code)
* [Spring Boot OAuth2 Part 2 - Getting The Access Token And Using it to Fetch Data.](https://www.javainuse.com/spring/spring-boot-oauth-access-token)
* [JBoss Drools Hello World-Stateful Knowledge Session using KieSession](https://www.javainuse.com/drools_hello_kie)
* [Understand Drools Stateful vs Stateless Knowledge Session](https://www.javainuse.com/drools_states)
* [JBoss Drools- Understanding Drools Decision Table using Simple Example](https://www.javainuse.com/drools/drools_decision)

See Also

* [Spring Boot Interview Questions](https://www.javainuse.com/spring/SpringBootInterviewQuestions)
* [Spring Batch Interview Questions](https://www.javainuse.com/spring/sprbatch_interview)
* [Spring AOP Interview Questions](https://www.javainuse.com/spring/spring-AOP-interview-quesions)
* [Angular 2 Interview Questions](https://www.javainuse.com/angular/ang2_intvw)
* [Apache Camel Interview Questions](https://www.javainuse.com/camel/Apache_Camel_Questions)
* [JBoss Fuse Interview Questions](https://www.javainuse.com/camel/JBoss_Fuse_Questions)
* [Drools Interview Questions](https://www.javainuse.com/drools/drools_intvw)
* [Java 8 Interview Questions](https://www.javainuse.com/java/java8_intvw)
* [Spring Cloud Interview Questions](https://www.javainuse.com/spring/spring-cloud-interview-questions)
* [Microservices Interview Questions](https://www.javainuse.com/spring/microservices-interview-quesions)
* [Java HashMap and ConcurrentHashMap Interview Questions](https://www.javainuse.com/java/java_map_intvw)
* [Mule ESB frequently asked interview questions](https://www.javainuse.com/misc/muleintvw)
* [Apache Kafka Interview Questions](https://www.javainuse.com/misc/apache-kafka-interview-questions)
* [Tosca Testing Tool Interview Questions](https://www.javainuse.com/misc/tosca-testing-tool-interview-questions)
* [Top Maven Build Tool Interview Questions](https://www.javainuse.com/misc/maven-interview-questions)
* [Top Gradle Build Tool Interview Questions](https://www.javainuse.com/misc/gradle-interview-questions)
* [Miscellaneous Topics](https://www.javainuse.com/misc)

© Copyright JavaInUse. All Rights Reserved.  
[Privacy Policy](https://www.javainuse.com/privacy)