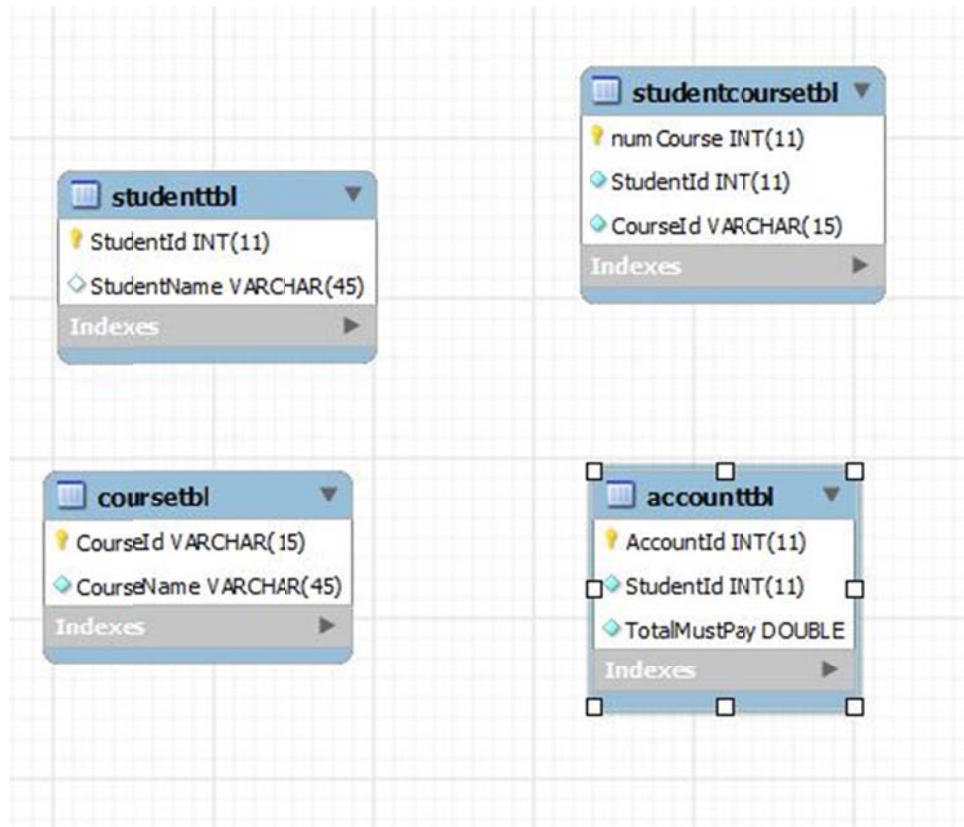


Transaction Example



```
public interface UniversalService {  
    public void insertStudentAndInsertStudentCourse(String studentName, String courseName) throws InvalidCourseId;  
}
```

Ex1:

Not use transaction annotation.

```

@Repository("universalServiceImpl")
public class UniversalServiceImpl implements UniversalService{

    @Resource(name="studentDaoImpl")
    private StudentDAO studentDaoImpl;

    @Resource(name="studentCourseDAOimpl")
    private StudentCourseDAO studentCourseDAOimpl;

    //@Transactional()
    public void insertStudentAndInsertStudentCourse(String studentName,String courseId) throws InvalidCourseId {

        Student student = new Student();
        student.setStudentName(studentName);

        student = studentDaoImpl.insertStudent(student);

        StudentCourse studentCourse = new StudentCourse();
        studentCourse.setStudentId(student.getStudentId());

        if ((!(courseId.equals("CS100"))) && (!(courseId.equals("CS200"))))
            throw new InvalidCourseId("Invalid CourseId Exception");

        studentCourse.setCourseId(courseId);

        studentCourseDAOimpl.insertStudentCourse(studentCourse);

    }

}

```

Run this test

```

@Configuration("app-context.xml")
@RunWith(SpringJUnit4ClassRunner.class)
public class TestUniversalServiceImpl {

    @Resource(name="universalServiceImpl")
    private UniversalService universalServiceImpl;

    @Test(expected = InvalidCourseId.class)
    public void testInsertStudent() throws InvalidCourseId{

        universalServiceImpl.insertStudentAndInsertStudentCourse("Tony", "S100");
        int i=1;
        assertTrue(i==1);
    }

}

```

Result (S100 → exception throws (should be CS100))

Filter:		↔	Edit
	StudentId	StudentName	
▶	5	Tony	
*	NULL	NULL	

Filter:		↔	Edit:	
	numCourse	StudentId	CourseId	
*	NULL	NULL	NULL	

➔ No roll back happens. After insert student to Student table. Exception happens and stops. And data still in table student.

Use transaction annotation with default configuration.

```
@Transactional
public void insertStudentAndInsertStudentCourse(String studentName,String courseId) throws InvalidCourseId {

    Student student = new Student();
    student.setStudentName(studentName);

    student = studentDaoImpl.insertStudent(student);

    StudentCourse studentCourse = new StudentCourse();
    studentCourse.setStudentId(student.getStudentId());

    if ((!(courseId.equals("CS100"))) && (!(courseId.equals("CS200"))))
        throw new InvalidCourseId("Invalid CourseId Exception");

    studentCourse.setCourseId(courseId);

    studentCourseDAOImpl.insertStudentCourse(studentCourse);

}
```

Run test and Result

Filter:		↔	Edit:		File:		Autosize:	
	StudentId	StudentName						
▶	6	Tony						
*	NULL	NULL						

Filter:		↔	Edit:	
	numCourse	StudentId	CourseId	
*	NULL	NULL	NULL	

Explain:

❖ **Warning:** Checked exceptions will not result in rollback by default

Use transaction annotation with twist configuration.

```
@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})
public void insertStudentAndInsertStudentCourse(String studentName,String courseId) throws InvalidCourseId {

    Student student = new Student();
    student.setStudentName(studentName);

    student = studentDaoImpl.insertStudent(student);

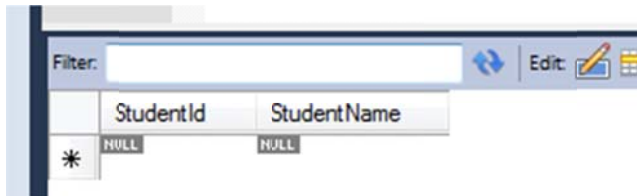
    StudentCourse studentCourse = new StudentCourse();
    studentCourse.setStudentId(student.getStudentid());

    if (((!(courseId.equals("CS100"))) && (!(courseId.equals("CS200")))))
        throw new InvalidCourseId("Invalid CourseId Exception");

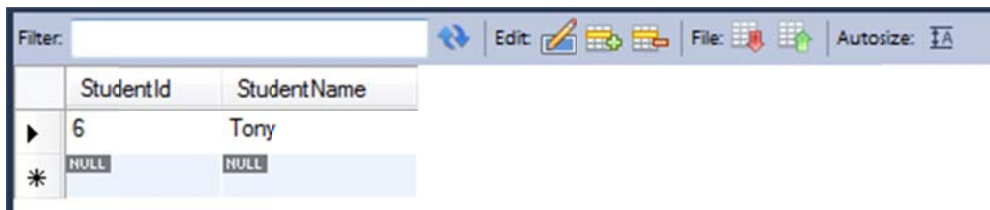
    studentCourse.setCourseId(courseId);

    studentCourseDAOimpl.insertStudentCourse(studentCourse);

}
```



	StudentId	StudentName
*	NULL	NULL



	StudentId	StudentName
▶	6	Tony
*	NULL	NULL

➔ Roll back happens when exception throws

Ex1:

Propagation Rules

➤ REQUIRED (Default)

- Execute method as part of current transaction or create a new transaction if one doesn't already exist

```
@Resource(name="studentDaoImpl")
private StudentDAO studentDaoImpl;

@Resource(name="studentCourseDAOimpl")
private StudentCourseDAO studentCourseDAOimpl;

@Resource(name="courseDAOimpl")
private AccountDAO courseDAOimpl;

@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})
public Student insertStudentAndInsertStudentCourse(String studentName,String courseId) throws InvalidCourseId {

    Student student = new Student();
    student.setStudentName(studentName);

    student = studentDaoImpl.insertStudent(student);

    StudentCourse studentCourse = new StudentCourse();
    studentCourse.setStudentId(student.getStudentid());

    if (!(courseId.equals("CS100"))) && !(courseId.equals("CS200"))))
        throw new InvalidCourseId("Invalid CourseId Exception");

    studentCourse.setCourseId(courseId);

    studentCourseDAOimpl.insertStudentCourse(studentCourse);

    return student;

}

@Transactional
public void TransactionInsertAccount(Account account) {
    courseDAOimpl.insertAccount(account);
}
```

Method `TransactionInsertStudentCourseAccount` uses 2 transactions

```
@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})
public Student insertStudentAndInsertStudentCourse(String studentName,String courseId)
```



```

@Transactional
public void TransactionInsertAccount(Account account) {
    courseDAOimpl.insertAccount(account);
}

```

→

create a new transaction if one doesn't already exist

→ We have 2 transaction separately

```

@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})
→ public Student insertStudentAndInsertStudentCourse(String studentName,String courseId)

@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})
→ public Student insertStudentAndInsertStudentCourse(String studentName,String courseId)

```

No use annotation transaction for TransactionInsertStudentCourseAccount

```

//@Transactional
public void TransactionInsertStudentCourseAccount(String studentName, String courseId,
    Double totalMustPay, boolean signalExceptionForTest) throws ExceptionToTest {

    Student student = new Student();
    try{
        student = insertStudentAndInsertStudentCourse(studentName, courseId);
    }
    catch (InvalidCourseId e){}

    if (signalExceptionForTest==true)
        throw new ExceptionToTest("Exception to test");

    Account account = new Account();
    account.setStudentId(student.getStudentid());
    account.setTotalMustPay(totalMustPay);

    TransactionInsertAccount(account);
}

```

Run Test

```

@Test
public void testTransactionInsertStudentCourseAccount() throws ExceptionToTest{
    universalServiceImpl.TransactionInsertStudentCourseAccount("Tony", "CS100", 1000.0, false);
    int i=1;
    assertTrue(i==1);
}

```

Result

Filter:		
	StudentId	StudentName
▶	12	Tony
*	NULL	NULL

Filter:			Edit:
	numCourse	StudentId	CourseId
▶	2	12	CS100
*	NULL	NULL	NULL

Filter:			Edit:
	AccountId	StudentId	TotalMustPay
▶	1	12	1000
*	NULL	NULL	NULL

Run Test

```

@Test
public void testTransactionInsertStudentCourseAccount() throws ExceptionToTest{
    universalServiceImpl.TransactionInsertStudentCourseAccount("Tony", "CS100", 1000.0, true);
    int i=1;
    assertTrue(i==1);
}

```

Result

Filter:		
	StudentId	StudentName
▶	13	Tony
*	NULL	NULL

Filter:			
	numCourse	StudentId	CourseId
▶	3	13	CS100
*	NULL	NULL	NULL

Filter:			
	AccountId	StudentId	TotalMustPay
*	NULL	NULL	NULL

Explain:

create a new transaction if one doesn't already exist

→ We have 2 transaction separately

```
@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})
→ public Student insertStudentAndInsertStudentCourse(String studentName,String courseId)

@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})
→ public Student insertStudentAndInsertStudentCourse(String studentName,String courseId)
```

The transaction below is done

```
@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})
public Student insertStudentAndInsertStudentCourse(String studentName,String courseId)
```

And exception is thrown and stopped. → transaction 1 is commit and transaction 2 not occur

```
@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})
public Student insertStudentAndInsertStudentCourse(String studentName,String courseId)
```


Ex2:

Propagation Rules

➤ REQUIRED (Default)

- Execute method as part of current transaction or create a new transaction if one doesn't already exist

use annotation transaction for TransactionInsertStudentCourseAccount

```
@Transactional(rollbackFor={com.dn.exceptions.ExceptionToTest.class})
public void TransactionInsertStudentCourseAccount(String studentName, String courseId,
    Double totalMustPay, boolean signalExceptionForTest) throws ExceptionToTest {

    Student student = new Student();
    try{
        student = insertStudentAndInsertStudentCourse(studentName, courseId);
    }
    catch (InvalidCourseId e){}

    if (signalExceptionForTest==true)
        throw new ExceptionToTest("Exception to test");

    Account account = new Account();
    account.setStudentId(student.getStudentid());
    account.setTotalMustPay(totalMustPay);

    TransactionInsertAccount(account);

}
```

Run the test

```

@Test
public void testTransactionInsertStudentCourseAccount() throws ExceptionToTest{

    universalServiceImpl.TransactionInsertStudentCourseAccount("Tony", "CS100", 1000.0, true);
    int i=1;
    assertTrue(i==1);
}

```

Result

Filter:		↔	Edit
	StudentId	StudentName	
*	NULL	NULL	

Filter:		↔	Edit
	numCourse	StudentId	CourseId
*	NULL	NULL	NULL

Filter:		↔	Edit
	AccountId	StudentId	TotalMustPay
*	NULL	NULL	NULL

Explain:

We use annotation transaction

```

@Transactional(rollbackFor={com.dn.exceptions.ExceptionToTest.class})
public void TransactionInsertStudentCourseAccount(String studentName, String courseId,
    Double totalMustPay, boolean signalExceptionForTest) throws ExceptionToTest {

```

- ➔ We have a transaction
- This transaction called 2 transactions.
- ➔

Propagation Rules

➤ REQUIRED (Default)

- Execute method as part of current transaction or create a new transaction if one doesn't already exist

➔ Execute method as part of current transaction

➔ We combine everything into 1 transaction.

After

```
@Transactional(rollbackFor={com.dn.exceptions.InvalidCourseId.class})  
public Student insertStudentAndInsertStudentCourse(String studentName,String courseId)
```

Is done.

And exception is thrown and stopped. → transaction 1 is rolled back and transaction 2 not occur