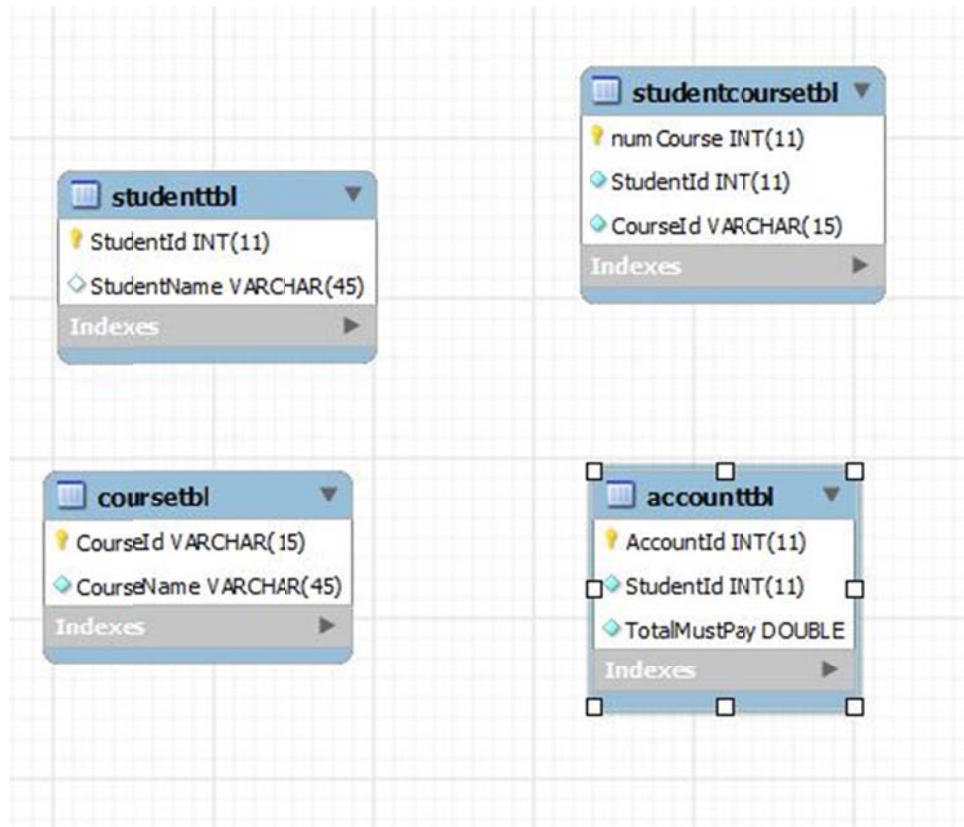# Transaction Example


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

Ex1:

Not use transaction annotation.

```java
@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

```java
@ContextConfiguration("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))

	StudentId	StudentName
▶	5	Tony
*	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.

```java
@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

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);


}
```
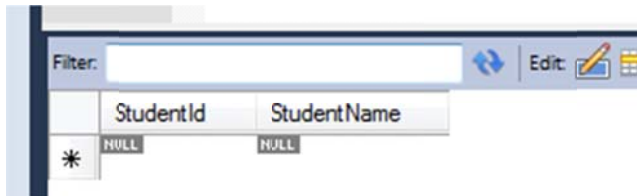
	StudentId	StudentName
*	NULL	NULL

	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

```java
@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);

}
```
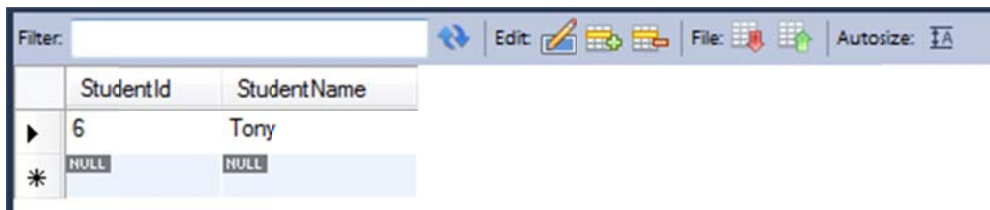
Method `TransactionInsertStudentCourseAccount` uses 2 transactions

```java
@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

	StudentId	StudentName
▶	12	Tony
*	NULL	NULL

	numCourse	StudentId	CourseId
▶	2	12	CS100
*	NULL	NULL	NULL

	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: [   ] ↻ Edit ✎

	StudentId	StudentName
▶	13	Tony
*	NULL	NULL

Filter: [   ] ↻ Edit ✎ 🗔 🗔

	numCourse	StudentId	CourseId
▶	3	13	C$100
*	NULL	NULL	NULL

Filter: [   ] ↻ Edit ✎ 🗔 🗔

	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`

```java
@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

```java
@Test
public void testTransactionInsertStudentCourseAccount() throws ExceptionToTest{

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

Result

	StudentId	StudentName
*	NULL	NULL

	numCourse	StudentId	CourseId
*	NULL	NULL	NULL

	AccountId	StudentId	TotalMustPay
*	NULL	NULL	NULL

Explain:

We use annotation transaction

```java
@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