

## 1. Transaction의 개념

- 1) 논리적 단위로 어떤 한 부분의 작업이 완료되었다 하더라도, 다른 부분의 작업이 완료되지 않을 경우 전체 취소되는 것이다.
- 2) 이때, 작업이 완료되는 것을 커밋(commit)이라고 하고, 작업이 취소되는 것을 롤백(rollback)이라고 한다.
- 3) 우리 일상생활에 transaction의 예는 많이 볼 수 있다.
- 4) 영화 예매를 할 경우 카드 결제 작업과 마일리지 적립 작업은 transaction으로 작동해야 한다.
- 5) 또한 은행 ATM기도 마찬가지 이다.
- 6) transaction은 그래서 성공적으로 처리되거나 또는 하나라도 실패하면 완전히 실패 처리를 해야 하는 경우에 사용된다.
- 7) Spring이 지원하는 transaction 방법 - 코드 기반 처리, 선언적 transaction, annotation 기반등을 사용한다.
- 8) 인터넷에서 도서를 구매할 경우 다음과 같은 순서가 필요할 것이다.
  - 결제 수행
  - 결제 내역 저장
  - 구매 내역 저장
- 9) 위의 과정 모두 성공적으로 이루어져야 한다.
- 10) 하나라도 실패할 경우 반드시 모든 과정이 취소되어야 한다.
- 11) 예를 들어, 결제 내역 저장까지는 성공했는데, 구매 내역을 저장하는 과정이 실패했다고 하자.
- 12) 이때, 전 과정이 취소되지 않는다면 구매자는 결제만 하고 구매는 하지 않은 것처럼 될 것이다.
- 13) 이처럼 transaction은 여러 과정을 하나의 행위로 묶을 때 사용된다.
- 14) transaction은 transaction 범위 내에 있는 처리 과정 중 한 가지라도 실패할 경우 전체 과정을 취소시킴으로써 데이터의 무결성을 보장한다.
- 15) 즉, transaction은 모두 반영하거나 모두 반영하지 않는 all or nothing 방식을 취한다.
- 16) transaction은 보통 4가지 특징인 ACID를 이용한다.
  - 원자성(Atomicity)
    - transaction은 한 개 이상의 동작을 논리적으로 한 개의 작업 단위(unit or work)로 묶는다.
    - 원자성은 transaction 범위에 있는 모든 동작이 모두 실행되거나 또는 모두 실행이 취소됨을 보장한다.
    - 모든 동작이 성공적으로 실행되면 transaction은 성공한다.
    - 만약 하나라도 실패하면 transaction은 실패하고 모든 과정을 롤백한다.
  - 일관성(Consistency)
    - transaction이 종료되면, system은 business에서 기대하는 상태가 된다.
    - 예를 들어, 서적 구매 transaction이 성공적으로 실행되면 결제 내역, 구매 내역, 잔고 정보가 business에 맞게 저장되고 변경된다.
  - 고립성(Isolation)
    - transaction은 다른 transaction과 독립적으로 실행되어야 하며, 서로 다른 transaction이 동일한 데이터에 동시에 접근할 경우 알맞게 동시 접근을 제어해야 한다.
  - 지속성(Durability)
    - transaction이 완료되면, 그 결과는 지속적으로 유지되어야 한다.
    - 현재의 application이 변경되거나 없어지더라도 data는 유지된다.
    - 일반적으로 물리적인 저장소를 통해서 transaction 결과가 저장된다.

## 2. Spring Transaction을 사용하지 않았을 경우

- transaction 처리를 하지 않았을 경우 rollback이 되지 않는 경우이다.

## 1) Spring Transaction Project 생성

- Package Explorer > right-click > New > Spring Legacy Project
- Project name : SpringTransactionDemo
- Select [Spring MVC Project]
- Next
- Project Settings - com.example.biz > Finish

## 2) Maven Install and Update

```
51 -In pom.xml > Dependencies tab > click [Add]
52 -[Enter groupId, artifactId or sha1... 'spring jdbc' 입력
53 -org.springframework spring-jdbc 선택
54 -[OK]
55 -pom.xml > right-click > Run As > Maven clean and Maven install
56
57 3)src/main/resource/oracle.properties 생성
58
59 db.driver=oracle.jdbc.driver.OracleDriver
60 db.url=jdbc:oracle:thin:@192.168.56.2:1521:ORCL
61 db.username=scott
62 db.password=tiger
63
64 4)pom.xml에 코드 추가
65
66 <dependency>
67 <groupId>com.oracle</groupId>
68 <artifactId>ojdbc8</artifactId>
69 <version>12.2</version>
70 </dependency>
71
72 5)servlet-context.xml 코드 추가
73
74 <context:component-scan base-package="com.example" />
75
76 <context:property-placeholder location="classpath:oracle.properties" />
77 <beans:bean id="dataSource"
78 class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
79 <beans:property name="driverClass" value="${db.driver}" />
80 <beans:property name="url" value="${db.url}" />
81 <beans:property name="username" value="${db.username}" />
82 <beans:property name="password" value="${db.password}" />
83 </beans:bean>
84
85 <beans:bean name="template"
86 class="org.springframework.jdbc.core.JdbcTemplate">
87 <beans:property name="dataSource" ref="dataSource" />
88 </beans:bean>
89
90 6)src/main/java/com.example.biz 이름변경
91 -com.example.controller
92
93 7)com.example.controller.HomeController.java
94
95 @Controller
96 public class HomeController {
97
98 private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
99
100 @RequestMapping(value = "/", method = RequestMethod.GET)
101 public String home(Model model) {
102 model.addAttribute("greeting", "Hello Spring Transaction");
103 return "home";
104 }
```

```
105     }
106
107 8)views/home.jsp
108
109     <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
110     <%@ page session="false" %>
111     <html>
112     <head>
113         <title>Home</title>
114     </head>
115     <body>
116         <h1>${greeting}</h1>
117     </body>
118     </html>
119
120 9)project > right-click > Run As > Run on server
121
122 10)Database Table 생성
123
124     CREATE TABLE Card
125     (
126         consumerId    VARCHAR2(20),
127         amount        NUMBER(2),
128         CONSTRAINT card_consumerid_pk PRIMARY KEY(consumerId)
129     );
130
131     CREATE TABLE Ticket
132     (
133         consumerId    VARCHAR2(20),
134         countnum      NUMBER(2),
135         CONSTRAINT ticket_consumerid_pk PRIMARY KEY(consumerId),
136         CONSTRAINT ticket_consumerid_fk FOREIGN KEY(consumerId) REFERENCES
            Card(consumerId),
137         CONSTRAINT ticket_countnum_ck CHECK(countnum < 5)
138     );
139
140 11)src/main/java/com.example.vo.TicketVO.java
141
142     package com.example.vo;
143
144     public class TicketVO {
145         private String consumerId;
146         private String amount;
147
148         public String getConsumerId() {
149             return consumerId;
150         }
151         public void setConsumerId(String consumerId) {
152             this.consumerId = consumerId;
153         }
154         public String getAmount() {
155             return amount;
156         }
157         public void setAmount(String amount) {
```

```
158         this.amount = amount;
159     }
160 }
161
162 12)src/main/java/com.example.dao.TicketDao.java
163
164 package com.example.dao;
165
166 import org.slf4j.Logger;
167 import org.slf4j.LoggerFactory;
168 import org.springframework.beans.factory.annotation.Autowired;
169 import org.springframework.jdbc.core.JdbcTemplate;
170 import org.springframework.stereotype.Repository;
171
172 import com.example.vo.TicketVO;
173
174 @Repository("ticketDao")
175 public class TicketDao {
176     @Autowired
177     private JdbcTemplate jdbcTemplate;
178
179     private static final Logger logger = LoggerFactory.getLogger(TicketDao.class);
180
181     public void buyTicket(final TicketVO ticketVO) {
182         logger.warn("buyTicket()");
183         logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
184         logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
185
186         String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
187         this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
188
189         sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
190         this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
191     }
192 }
193
194 13)views/buy_ticket.jsp
195
196 <%@ page language="java" contentType="text/html; charset=UTF-8"
197     pageEncoding="UTF-8"%>
198 <!DOCTYPE html>
199 <html>
200     <head>
201         <meta charset="UTF-8">
202         <title>Ticket 구매 창</title>
203     </head>
204     <body>
205         <h1>카드 결제</h1>
206
207         <form action="buy_ticket_card" method="post">
208             고객 아이디 : <input type="text" name="consumerId" > <br />
209             티켓 구매수 : <input type="text" name="amount" > <br />
210             <input type="submit" value="구매" >
211         </form>
```

```
212
213     </body>
214 </html>
215
216 14)views/buy_ticket_result.jsp
217
218     <%@ page language="java" contentType="text/html; charset=UTF-8"
219         pageEncoding="UTF-8"%>
220     <!DOCTYPE html>
221     <html>
222     <head>
223     <meta charset="UTF-8">
224     <title>Ticket 구매 결과 창</title>
225     </head>
226     <body>
227     <h1>Ticket 구매 결과</h1>
228     <ul>
229     <li>고객 아이디 : ${ticketInfo.consumerId }</li>
230     <li>구매 갯수 : ${ticketInfo.amount }</li>
231     </ul>
232     </body>
233 </html>
234
235 15)HomeController.java 코드 추가
236
237     package com.javasoft.biz;
238
239     import org.springframework.beans.factory.annotation.Autowired;
240     import org.springframework.stereotype.Controller;
241     import org.springframework.ui.Model;
242     import org.springframework.web.bind.annotation.RequestMapping;
243     import org.springframework.web.bind.annotation.RequestMethod;
244
245     @Controller
246     public class HomeController {
247         @Autowired
248         private TicketDao dao;
249
250         @RequestMapping(value = "/", method = RequestMethod.GET)
251         public String home(Model model) {
252             model.addAttribute("greeting", "Hello Spring Transaction");
253             return "home";
254         }
255
256         @RequestMapping("/buy_ticket")
257         public String buy_ticket() {
258             return "buy_ticket";
259         }
260     }
261
262     @RequestMapping(value = "/buy_ticket_card", method = RequestMethod.POST)
263     public String buy_ticket_card(TicketVO ticketVO, Model model) {
264         logger.warn( "buy_ticket_card" );
265         logger.warn( "고객 아이디 : " + ticketVO.getConsumerId() );
```

```
266         logger.warn( "구매 갯수 : " + ticketVO.getAmount() );
267
268         dao.buyTicket(ticketVO);
269
270         model.addAttribute("ticketInfo", ticketVO);
271
272         return "buy_ticket_result";
273     }
274 }
```

275  
276 16)project > right-click > Run As > Run on server

277  
278 17)[http://localhost:8080/biz/buy\\_ticket](http://localhost:8080/biz/buy_ticket)

279 -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.

280 -하지만 구매 갯수를 5장 이상을 입력하면 오류 발생

281 --Card table에는 구매 갯수가 5 이상의 값 입력이 가능하다.

282 --하지만 Ticket table에는 ORA-02290: check constraint (SCOTT.TICKET\_COUNTNUM\_CHK) 위반  
이 발생했기 때문에 입력에 실패하게 된다.

283 -이렇게 오류가 나오는 것이 정상이다.

284 -그리고 치명적인 오류인 것은 Car 테이블에는 5장 이상 구매한 데이터는 입력이 되지만, Ticket table에는 체크 위반  
때문에 입력되지 않는다는 것이다.

285 -현재 Card table에는 5장 이상이 입력된 상태이지만, Ticket table에는 에러가 발생했기 때문에 5장 이상이 입력되  
어 있지 않다.

286 -두 테이블 모두 입력 성공하거나 입력 취소가 되어야 한다.

### 287 288 289 3. Spring Transaction 처리

#### 290 1)JDBC 기반 Transaction Manager 설정

291 -JDBC나 MyBatis와 같이 JDBC를 이용하는 database 연동을 처리하는 경우

```
292
293 <bean name="transactionManager"
294       class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
295     <property name="dataSource" ref="dataSource" />
296   </bean>
```

#### 297 2)JPA Transaction Manager 설정

298 -JPA를 사용할 경우

```
299
300 <bean id="entityManagerFactory"
301       class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
302     ...
303   </bean>
304
305 <bean name="transactionManager"
306       class="org.springframework.orm.jpa.JpaTransactionManager">
307     <property name="entityManagerFactory" ref="entityManagerFactory" />
308   </bean>
```

#### 308 3)Hibernate Transaction Manager 설정

309 -Hibernate를 사용하는 경우

```
310
311 <bean id="transactionManager"
312       class="org.springframework.orm.hibernate3.HibernateTransactionManager">
313     <property name="sessionFactory" ref="sessionFactory" />
```

```
313     </bean>
314
315     <bean id="sessionFactory"
316           class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
317         <property name="dataSource" ref="dataSource" />
318         ...
319     </bean>
320
321 4)Transaction 전파 속성
322 -org.springframework.transaction.TransactionDefinition interface
323 -2개 이상의 transaction이 작동할 때, 기존의 transaction에 참여하는 방법을 결정하는 속성.
324 -PROPAGATION_REQUIRED : 0
325     --method를 수행하는 데 transaction이 필요하다는 것을 의미
326     --현재 진행 중인 transaction이 존재하면, 해당 transaction을 사용한다.
327     --존재하지 않는다면 새로운 transaction을 생성한다.
328     --default, 즉 전체 처리한다.
329
330 -PROPAGATION_MANDATORY : 2
331     --method를 수행하는 데 transaction이 필요하다는 것을 의미한다.
332     --하지만 위의 REQUIRED와 달리, 진행중인 transaction이 존재하지 않을 경우 exception을 발생시킴
333     --transaction에 꼭 포함되어야 함
334
335 -PROPAGATION_REQUIRES_NEW : 3
336     --항상 새로운 transaction을 시작한다.
337     --기존 transaction이 존재하면 기존 transaction을 일시 중지하고 새로운 transaction을 시작한다.
338     --새로 시작된 transaction이 종료된 뒤에 기존 transaction이 계속된다.
339     --각자 transaction 처리(별도의 transaction 처리)
340
341 -PROPAGATION_SUPPORTS : 1
342     --method가 transaction을 필요로 하지 않지만, 기존 transaction이 존재할 경우 transaction을 사용한다는
343     것을 의미한다.
344     --진행중인 transaction이 존재하지 않더라도 method는 정상적으로 동작한다.
345     --기존 transaction에 의존
346
347 -PROPAGATION_NOT_SUPPORTED : 4
348     --method가 transaction을 필요로 하지 않음을 의미한다.
349     --SUPPORTS와 달리 진행 중인 transaction이 존재할 경우 method가 실행되는 동안 transaction은 일시 중
350     지되며, method 실행이 종료된 뒤에 transaction을 계속 진행한다.
351     --transaction에 포함하지 않음 -> transaction이 없는 것과 동일
352
353 -PROPAGATION_NEVER : 5
354     --method가 transaction을 필요로 하지 않으며, 만약 진행 중인 transaction이 존재하면 exception을 발생시
355     킴.
356     --transaction에 절대 포함하지 않음.
357
358 -PROPAGATION_NESTED : 6
359     --기존 transaction이 존재하면, 기존 transaction에 중첩된 transaction에서 method를 실행한다.
360     --기존 transaction이 존재하지 않으면 REQUIRED와 동일하게 동작한다.
361     --이 기능은 JDBC 3.0 driver를 사용할 때에만 적용된다.
362
363 REQUIRED <----> REQUIRES_NEW
364 MANDATORY <---> NEVER
365 SUPPORTS <---> NOT_SUPPORTED
```

```

363 5)Spring에서 설정 가능한 transaction 격리 level
364 -ISOLATION_DEFAULT
365 --기본 설정 사용
366
367 -ISOLATION_READ_UNCOMMITTED
368 --다른 transaction에서 commit하지 않은 data를 읽을 수 있다.
369
370 -ISOLATION_READ_COMMITTED
371 --다른 transaction에 의해 commit된 data를 읽을 수 있다.
372
373 -ISOLATION_REPEATABLE_READ
374 --처음에 읽어 온 data와 두 번째 읽어 온 data가 동일한 값을 갖는다.
375
376 -ISOLATION_SERIALIZABLE
377 --동일한 data에 대해서 동시에 두 개 이상의 transaction이 수행될 수 있다.
378
379
380 4. PlatformTransactionManager를 사용하는 Lab
381 1)applicationContext.xml에 다음의 코드를 추가한다.
382
383 <beans:bean name="transactionManager"
384 class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
385 <beans:property name="dataSource" ref="dataSource" />
386 </beans:bean>
387
388 2)TicketDao.java 코드 추가
389
390 @Autowired
391 private PlatformTransactionManager transactionManager; <-- 코드 추가
392
393 public void buyTicket(final TicketVO ticketVO) {
394     logger.warn("buyTicket()");
395     logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
396     logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
397
398     TransactionDefinition definition = new DefaultTransactionDefinition();
399     <--코드 추가
400     TransactionStatus status = this.transactionManager.getTransaction(definition); <--
401     코드 추가
402
403     try {
404         <--코드 추가
405         String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
406         this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
407
408         sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
409         this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
410         this.transactionManager.commit(status);
411         <--코드 추가
412     }catch(Exception e) {
413         <--코드 추가
414         e.printStackTrace();
415         <--코드 추가
416         this.transactionManager.rollback(status);

```



```

410         <--코드 추가
411     }
412     <--코드 추가
413 }
414
415 3)project > right-click > Run As > Run on server
416
417 4)http://localhost:8080/biz/buy\_ticket
418     -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
419     -하지만 구매 갯수를 5장 이상을 입력하면 두 개의 테이블 모두 입력되지 않음.
420     -transaction 성공했음.
421
422 5. TransactionTemplate 이용한 Transaction 처리하기
423     1)기본적으로 사용한 PlatformTransactionManager interface보다 더 많이 사용되는 객체는
424     TransactionTemplate 이다.
425     2)Transaction을 처리하기 위해 PlatformTransactionManager의 메소드를 직접 사용해도 되지만 try/catch 블
426     록을 써야 하는 번거로움이 발생한다.
427     3)Transaction 안에서 작업 중에 예외가 발생한 경우에는 Transaction을 롤백해주도록 만들어야 하기 때문이다.
428     4)그래서 PlatformTransactionManager의 메소드를 직접 사용하는 대신 Template/Callback 방식의
429     TransactionTemplate을 이용하면 편리하다.
430
431 5)applicationContext.xml 코드 추가
432
433     <beans:bean name="transactionTemplate"
434     class="org.springframework.transaction.support.TransactionTemplate">
435         <beans:property name="transactionManager" ref="transactionManager" />
436     </beans:bean>
437
438 6)TicketDao.java 코드 수정 및 추가
439
440     @Repository("ticketDao")
441     public class TicketDao {
442         @Autowired
443         private JdbcTemplate jdbcTemplate;
444
445         @Autowired
446         private TransactionTemplate transactionTemplate;
447
448         public void buyTicket(final TicketVO ticketVO) {
449             logger.warn("buyTicket()");
450             logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
451             logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
452
453             this.transactionTemplate.execute(new TransactionCallbackWithoutResult() {
454
455                 @Override
456                 protected void doInTransactionWithoutResult(TransactionStatus status) {
457                     String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
458                     TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
459                     ticketVO.getAmount());
460                 }
461             });
462         }
463     }

```

```

457         sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
458         TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
        ticketVO.getAmount());
459         // 트랜잭션 안에서 동작하는 코드, 트랜잭션 매니저와 연결되어 있는 모든 DAO는 같은 트랜잭션에 참여한다.

460         // 정상적으로 작업을 마치고 리턴되면 트랜잭션은 커밋된다.
461         // 만약 이전에 시작한 트랜잭션에 참여했다면 해당 트랜잭션의 작업을 모두 마칠 때까지 커밋은 보류된다.
462         // 리턴되기 이전에 예외가 발생하면 트랜잭션은 롤백된다.
463     }
464
465     });
466 }
467 }

```

4)project > right-click > Run As > Run on server

5)[http://localhost:8080/biz/buy\\_ticket](http://localhost:8080/biz/buy_ticket)

- 구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
- 하지만 구매 갯수를 5장 이상을 입력하면 두 개의 테이블 모두 입력되지 않음.
- 다만 화면에 에러 메시지 display
- transaction 성공했음.

## 6. Transaction 전파에 관한 Lab

1)com.example.service.TicketService.java

```

480
481     package com.example.service;
482
483     import com.example.vo.TicketVO;
484
485     public interface TicketService {
486         void execute(TicketVO ticketVO);
487     }

```

2)com.example.service.TicketServiceImpl.java

```

490
491     package com.example.service;
492
493     import org.springframework.stereotype.Service;
494     import org.springframework.transaction.TransactionStatus;
495     import org.springframework.transaction.support.TransactionCallbackWithoutResult;
496     import org.springframework.transaction.support.TransactionTemplate;
497
498     import com.example.dao.TicketDao;
499     import com.example.vo.TicketVO;
500
501     @Service("ticketService")
502     public class TicketServiceImpl implements TicketService {
503         private TicketDao ticketDao;
504         private TransactionTemplate transactionTemplate2;
505
506         public void setTicketDao(TicketDao ticketDao) {
507             this.ticketDao = ticketDao;
508         }

```

```
509
510     public void setTransactionTemplate2(TransactionTemplate transactionTemplate2) {
511         this.transactionTemplate2 = transactionTemplate2;
512     }
513
514     @Override
515     public void execute(final TicketVO ticketVO) {
516         transactionTemplate2.execute(new TransactionCallbackWithoutResult() {
517
518             @Override
519             protected void doInTransactionWithoutResult(TransactionStatus status) {
520                 ticketVO.setAmount("1");
521                 ticketDao.buyTicket(ticketVO);
522             }
523         });
524     }
525 }
526 }
```

### 528 3)HomeController.java

```
529
530     @Autowired
531     private TicketService ticketService;
532     ...
533     ...
534
535     @RequestMapping(value = "/buy_ticket_card", method = RequestMethod.POST)
536     public String buy_ticket_card(TicketVO ticketVO, Model model) {
537         logger.warn( "buy_ticket_card" );
538         logger.warn( "고객 아이디 : " + ticketVO.getConsumerId() );
539         logger.warn( "구매 갯수 : " + ticketVO.getAmount() );
540
541         //dao.buyTicket(ticketVO);
542         this.ticketService.execute(ticketVO);
543         model.addAttribute("ticketInfo", ticketVO);
544
545         return "buy_ticket_result";
546     }
547 }
```

### 548 4)TicketDao.java

```
549
550     package com.example.dao;
551
552     import org.slf4j.Logger;
553     import org.slf4j.LoggerFactory;
554     import org.springframework.beans.factory.annotation.Autowired;
555     import org.springframework.jdbc.core.JdbcTemplate;
556     import org.springframework.stereotype.Repository;
557     import org.springframework.transaction.TransactionStatus;
558     import org.springframework.transaction.support.TransactionCallbackWithoutResult;
559     import org.springframework.transaction.support.TransactionTemplate;
560
561     import com.example.vo.TicketVO;
562
```

```
563     @Repository("ticketDao")
564     public class TicketDao {
565         private JdbcTemplate jdbcTemplate;
566         private TransactionTemplate transactionTemplate1;
567
568         private static final Logger logger = LoggerFactory.getLogger(TicketDao.class);
569
570         public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {
571             this.jdbcTemplate = jdbcTemplate;
572         }
573
574         public void setTransactionTemplate1(TransactionTemplate transactionTemplate1) {
575             this.transactionTemplate1 = transactionTemplate1;
576         }
577
578         public void buyTicket(final TicketVO ticketVO) {
579             logger.warn("buyTicket()");
580             logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
581             logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
582
583             this.transactionTemplate1.execute(new TransactionCallbackWithoutResult() {
584
585                 @Override
586                 protected void doInTransactionWithoutResult(TransactionStatus status) {
587                     String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
588                     TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
589                                                         ticketVO.getAmount());
590
591                     sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
592                     TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
593                                                         ticketVO.getAmount());
594                 }
595             });
596         }
597     }
```

598 5) servlet-context.xml

```
599
600     ...
601     <beans:bean name="transactionTemplate1"
602     class="org.springframework.transaction.support.TransactionTemplate">
603         <beans:property name="transactionManager" ref="transactionManager" />
604         <beans:property name="propagationBehavior" value="0" />
605     </beans:bean>
606     <beans:bean name="transactionTemplate2"
607     class="org.springframework.transaction.support.TransactionTemplate">
608         <beans:property name="transactionManager" ref="transactionManager" />
609         <beans:property name="propagationBehavior" value="0" />
610     </beans:bean>
611     <beans:bean name="ticketDao" class="com.example.dao.TicketDao" >
612         <beans:property name="jdbcTemplate" ref="template" />
613         <beans:property name="transactionTemplate1" ref="transactionTemplate1" />
614     </beans:bean>
```

```

613
614     <beans:bean name="ticketService" class="com.example.service.TicketServiceImpl" >
615         <beans:property name="ticketDao" ref="ticketDao" />
616         <beans:property name="transactionTemplate2" ref="transactionTemplate2" />
617     </beans:bean>
618
619
620 7. 선언적 Transaction 처리
621 1)Transaction처리를 code에서 직접 수행하지 않고, 설정 file이나 annotation을 이용해서 transaction의 범위,
    rollback 규칙 등을 정의하는 방식
622 2)두 가지 방식으로 정의가능
623     -<tx:advice> tag를 이용한 transaction
624     -@Transactional annotation을 이용한 transaction 설정
625
626 3)tx namespace를 이용한 Transaction 설정
627     -먼저 tx namespace를 추가해야 한다.
628     -tx namespace를 beans tag에 추가한 뒤, <tx:advice> tag, <tx:attribute> tag, 그리고
        <tx:method> tag를 이용해서 transaction 속성을 정의할 수 있다.
629     -<tx:advice> tag는 transaction을 적용할 때 사용될 Advisor를 생성한다.
630     -id속성은 생성될 transaction Advisor의 식별값을 입력하며, transaction-manager 속성에는 Spring의
        PlatformTransactionManager bean을 설정한다.
631     -<tx:method> tag의 속성
632         --name : transaction이 적용될 method 이름을 명시.
633         ---"*"을 사용한 설정이 가능.
634         ---예를 들어 "get*"으로 설정할 경우 이름이 get으로 시작하는 method를 의미.
635         --propagation : transaction 전파 규칙을 설정.
636         ---REQUIRED(기본값), MANDATORY, REQUIRES_NEW, SUPPORTS, NOT_SUPPORTED,
        NEVER, NVESTED
637         --isolation : transaction 격리 level을 설정
638         ---DEFAULT, READ, READ_UNCOMMITTED, READ_COMMITTED, REPEATABLE_READ,
        SERIALIZABLE
639         --read-only : 읽기 전용 여부를 설정.
640         --no-rollback-for : transaction을 rollback하지 않을 Exception Type을 지정.
641         --rollback-for : transaction을 rollback할 Exception type을 지정.
642         --timeout : transaction의 timeout 시간을 초 단위로 지정.
643     -<tx:advice> tag는 Advisor만 생성하는 것이지 실제로 transaction을 적용하는 것은 아니다.
644     -실제로 transaction을 적용하는 것은 AOP를 통해서 이루어진다.
645
646 4)Annotation 기반 Transaction 설정
647     -@Transactional annotation을 사용해서 transaction 설정 가능.
648     -@Transactional annotation 의 주요 속성
649         --propagation : transaction 전파 규칙 설정.
650         ---org.springframework.transaction.annotation.Propagation 나열형으로 정의돼 있음.
651         ---기본값은 Propagation.REQUIRED이다.
652         --isolation : transaction 격리 level 설정.
653         ---org.springframework.transaction.annotation.Isolation 나열형으로 정의돼 있음.
654         --readOnly : 읽기 전용 여부 설정. boolean 값을 설정하며, 기본값은 false이다.
655         --rollbackFor : transaction을 rollback할 Exception Type을 설정.
656         ---예 : rollbackFor={Exception.class}
657         --noRollbackFor : transaction을 rollback하지 않을 Exception Type 설정.
658         ---예 : noRollbackFor={ItemNotFoundException.class}
659         --timeout : transaction의 timeout 시간을 초 단위로 지정.
660     -@Transactional annotation이 적용될 Spring Bean에 Transaction을 실제로 적용하려면 다음과 같이
        <tx:annotation-driven>을 설정해야 한다.

```

```
661     <tx:annotation-driven transaction-manager="transactionManager" />
662 -<tx:annotation-driven> tag 속성
663 --transaction-manager : 사용할 PlatformTransaction Manager bean 이름.
664 --기본값은 transactionManager
665 --proxy-target-class : class에 대해서 proxy를 생성 여부.
666 ---true는 CGLIB를 이용해서 proxy를 생성
667 ---false는 Java dynamic proxy를 이용해서 proxy를 생성, 기본값
668 --order : Advice 적용 순서 : int의 최대값(가장 낮은 순위).
669
670 8. Lab : tx namespace를 이용한 Transaction 설정
671 1)Spring Transaction Project 생성
672 -Package Explorer > right-click > New >Spring Legacy Project
673 -Project name : SpringTransactionDemo
674 -Select [Spring MVC Project]
675 -Next
676 -Project Settings - com.example.biz > Finish
677
678 2)Maven Install and Update
679 -In pom.xml > Dependencies tab > click [Add]
680 -[Enter groupId, artifactId or sha1... 'spring jdbc' 입력
681 -org.springframework spring-jdbc 선택
682 -[OK]
683 -pom.xml > right-click > Run As > Maven clean and Maven install
684
685 3)src/main/resource/oracle.properties 생성
686
687 db.driver=oracle.jdbc.driver.OracleDriver
688 db.url=jdbc:oracle:thin:@192.168.56.2:1521:ORCL
689 db.username=scott
690 db.password=tiger
691
692 4)pom.xml에 코드 추가
693
694 <dependency>
695   <groupId>com.oracle</groupId>
696   <artifactId>ojdbc8</artifactId>
697   <version>12.2</version>
698 </dependency>
699
700 5)servlet-context.xml 코드 추가
701
702 <context:component-scan base-package="com.example" />
703
704 <context:property-placeholder location="classpath:oracle.properties" />
705 <beans:bean id="dataSource"
706   class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
707   <beans:property name="driverClass" value="${db.driver}" />
708   <beans:property name="url" value="${db.url}" />
709   <beans:property name="username" value="${db.username}" />
710   <beans:property name="password" value="${db.password}" />
711 </beans:bean>
712
713 <beans:bean name="template"
714   class="org.springframework.jdbc.core.JdbcTemplate">
```

```
715     <beans:property name="dataSource" ref="dataSource" />
716 </beans:bean>
717
718 6)src/main/java/com.example.biz 이름변경
719   -com.example.controller
720
721 7)com.example.controller.HomeController.java
722
723 @Controller
724 public class HomeController {
725
726     private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
727
728     @RequestMapping(value = "/", method = RequestMethod.GET)
729     public String home(Model model) {
730         model.addAttribute("greeting", "Hello Spring Transaction");
731         return "home";
732     }
733 }
734
735 8)views/home.jsp
736
737 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
738 <%@ page session="false" %>
739 <html>
740 <head>
741     <title>Home</title>
742 </head>
743 <body>
744     <h1>${greeting}</h1>
745 </body>
746 </html>
747
748 9)project > right-click > Run As > Run on server
749
750 10)Database Table 생성
751
752 CREATE TABLE Card
753 (
754     consumerId    VARCHAR2(20),
755     amount        NUMBER(2),
756     CONSTRAINT card_consumerid_pk PRIMARY KEY(consumerId)
757 );
758
759 CREATE TABLE Ticket
760 (
761     consumerId    VARCHAR2(20),
762     countnum      NUMBER(2),
763     CONSTRAINT ticket_consumerid_pk PRIMARY KEY(consumerId),
764     CONSTRAINT ticket_consumerid_fk FOREIGN KEY(consumerId) REFERENCES
       Card(consumerId),
765     CONSTRAINT ticket_countnum_ck CHECK(countnum < 5)
766 );
767
```

```
768 11)src/main/java/com.example.vo.TicketVO.java
769
770 package com.example.vo;
771
772 public class TicketVO {
773     private String consumerId;
774     private String amount;
775
776     public String getConsumerId() {
777         return consumerId;
778     }
779     public void setConsumerId(String consumerId) {
780         this.consumerId = consumerId;
781     }
782     public String getAmount() {
783         return amount;
784     }
785     public void setAmount(String amount) {
786         this.amount = amount;
787     }
788 }
789
790 12)src/main/java/com.example.dao.TicketDao.java
791
792 package com.example.dao;
793
794 import org.slf4j.Logger;
795 import org.slf4j.LoggerFactory;
796 import org.springframework.beans.factory.annotation.Autowired;
797 import org.springframework.jdbc.core.JdbcTemplate;
798 import org.springframework.stereotype.Repository;
799
800 import com.example.vo.TicketVO;
801
802 @Repository("ticketDao")
803 public class TicketDao {
804     @Autowired
805     private JdbcTemplate jdbcTemplate;
806
807     private static final Logger logger = LoggerFactory.getLogger(TicketDao.class);
808
809     public void buyTicket(final TicketVO ticketVO) {
810         logger.warn("buyTicket()");
811         logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
812         logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
813
814         String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
815         this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
816
817         sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
818         this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
819     }
820 }
821
```



```
822 13)views/buy_ticket.jsp
823
824 <%@ page language="java" contentType="text/html; charset=UTF-8"
825     pageEncoding="UTF-8"%>
826 <!DOCTYPE html>
827 <html>
828     <head>
829         <meta charset="UTF-8">
830         <title>Ticket 구매 창</title>
831     </head>
832     <body>
833         <h1>카드 결제</h1>
834
835         <form action="buy_ticket_card" method="post">
836             고객 아이디 : <input type="text" name="consumerId" > <br />
837             티켓 구매수 : <input type="text" name="amount" > <br />
838             <input type="submit" value="구매" >
839         </form>
840
841     </body>
842 </html>
843
```

```
844 14)views/buy_ticket_result.jsp
845
846 <%@ page language="java" contentType="text/html; charset=UTF-8"
847     pageEncoding="UTF-8"%>
848 <!DOCTYPE html>
849 <html>
850     <head>
851         <meta charset="UTF-8">
852         <title>Ticket 구매 결과 창</title>
853     </head>
854     <body>
855         <h1>Ticket 구매 결과</h1>
856         <ul>
857             <li>고객 아이디 : ${ticketInfo.consumerId }</li>
858             <li>구매 갯수 : ${ticketInfo.amount }</li>
859         </ul>
860     </body>
861 </html>
862
```

```
863 15)HomeController.java 코드 추가
864
865 package com.javasoft.biz;
866
867 import org.springframework.beans.factory.annotation.Autowired;
868 import org.springframework.stereotype.Controller;
869 import org.springframework.ui.Model;
870 import org.springframework.web.bind.annotation.RequestMapping;
871 import org.springframework.web.bind.annotation.RequestMethod;
872
873 @Controller
874 public class HomeController {
875     @Autowired
```

```

876     private TicketDao dao;
877
878     @RequestMapping(value = "/", method = RequestMethod.GET)
879     public String home(Model model) {
880         model.addAttribute("greeting", "Hello Spring Transaction");
881         return "home";
882     }
883
884     @RequestMapping("/buy_ticket")
885     public String buy_ticket() {
886         return "buy_ticket";
887     }
888
889     @RequestMapping(value = "/buy_ticket_card", method = RequestMethod.POST)
890     public String buy_ticket_card(TicketVO ticketVO, Model model) {
891         logger.warn( "buy_ticket_card" );
892         logger.warn( "고객 아이디 : " + ticketVO.getConsumerId() );
893         logger.warn( "구매 갯수 : " + ticketVO.getAmount() );
894
895         dao.buyTicket(ticketVO);
896
897         model.addAttribute("ticketInfo", ticketVO);
898
899         return "buy_ticket_result";
900     }
901 }
902 }
903

```

16)project > right-click > Run As > Run on server

17)[http://localhost:8080/biz/buy\\_ticket](http://localhost:8080/biz/buy_ticket)

- 구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
- 하지만 구매 갯수를 5장 이상을 입력하면 오류 발생
  - Card table에는 구매 갯수가 5 이상의 값 입력이 가능하다.
  - 하지만 Ticket table에는 ORA-02290: check constraint (SCOTT.TICKET\_COUNTNUM\_CHK) 위반 이 발생했기 때문에 입력에 실패하게 된다.
- 이렇게 오류가 나오는 것이 정상이다.
- 그리고 치명적인 오류인 것은 Car 테이블에는 5장 이상 구매한 데이터는 입력이 되지만, Ticket table에는 체크 위반 때문에 입력되지 않는다는 것이다.
- 현재 Card table에는 5장 이상이 입력된 상태이지만, Ticket table에는 에러가 발생했기 때문에 5장 이상이 입력되어 있지 않다.
- 두 테이블 모두 입력 성공하거나 입력 취소가 되어야 한다.

18)AspectJ Weaver library 설치

- Maven Repository에서 'aspectj weaver'으로 검색
- aspectj weaver 1.9.2 버전을 pom.xml에 추가

```

<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.9.2</version>
</dependency>

```

19)Spring AOP library 설치

```
927 -Maven Repository에서 'spring aop'으로 검색
928 -aspectj weaver 4.3.20 버전을 pom.xml에 추가
929
930     <dependency>
931         <groupId>org.springframework</groupId>
932         <artifactId>spring-aop</artifactId>
933         <version>4.3.20.RELEASE</version>
934     </dependency>
935
936 -Maven Install
937
938 20)servlet-context.xml에서
939 -aop, tx check
940 -applicationContext.xml에 다음의 코드를 추가한다.
941
942     <beans:bean name="transactionManager"
943         class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
944         <beans:property name="dataSource" ref="dataSource" />
945     </beans:bean>
946
947     <tx:advice id="txAdvice" transaction-manager="transactionManager">
948         <tx:attributes>
949             <tx:method name="buyTicket" propagation="REQUIRED"/>
950         </tx:attributes>
951     </tx:advice>
952
953     <aop:config>
954         <aop:pointcut expression="execution(public * com.example.dao.TicketDao.*(..))"
955             id="servicePublicMethod"/>
956         <aop:advisor advice-ref="txAdvice" pointcut-ref="servicePublicMethod"/>
957     </aop:config>
958
959 21)project > right-click > Run As > Run on server
960
961 22)http://localhost:8080/biz/buy\_ticket
962 -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
963 -하지만 구매 갯수를 5장 이상을 입력하면 두 개의 테이블 모두 입력되지 않음.
964 -다만 화면에 에러 메시지 display
965 -transaction 성공했음.
966
967 9. Lab : Annotation 기반 Transaction 설정
968 1)Spring Transaction Project 생성
969 -Package Explorer > right-click > New > Spring Legacy Project
970 -Project name : SpringTransactionDemo
971 -Select [Spring MVC Project]
972 -Next
973 -Project Settings - com.example.biz > Finish
974
975 2)Maven Install and Update
976 -In pom.xml > Dependencies tab > click [Add]
977 -[Enter groupId, artifactId or sha1... 'spring jdbc' 입력
978 -org.springframework spring-jdbc 선택
979 -[OK]
```

```
979 -pom.xml > right-click > Run As > Maven clean and Maven install
980
981 3)src/main/resource/oracle.properties 생성
982
983 db.driver=oracle.jdbc.driver.OracleDriver
984 db.url=jdbc:oracle:thin:@192.168.56.2:1521:ORCL
985 db.username=scott
986 db.password=tiger
987
988 4)pom.xml에 코드 추가
989
990 <dependency>
991 <groupId>com.oracle</groupId>
992 <artifactId>ojdbc8</artifactId>
993 <version>12.2</version>
994 </dependency>
995
996 5)servlet-context.xml 코드 추가
997
998 <context:component-scan base-package="com.example" />
999
1000 <context:property-placeholder location="classpath:oracle.properties" />
1001 <beans:bean id="dataSource"
1002 class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
1003 <beans:property name="driverClass" value="${db.driver}" />
1004 <beans:property name="url" value="${db.url}" />
1005 <beans:property name="username" value="${db.username}" />
1006 <beans:property name="password" value="${db.password}" />
1007 </beans:bean>
1008
1009 <beans:bean name="template"
1010 class="org.springframework.jdbc.core.JdbcTemplate">
1011 <beans:property name="dataSource" ref="dataSource" />
1012 </beans:bean>
1013
1014 6)src/main/java/com.example.biz 이름변경
1015 -com.example.controller
1016
1017 7)com.example.controller.HomeController.java
1018
1019 @Controller
1020 public class HomeController {
1021
1022 private static final Logger logger = LoggerFactory.getLogger(HomeController.class);
1023
1024 @RequestMapping(value = "/", method = RequestMethod.GET)
1025 public String home(Model model) {
1026 model.addAttribute("greeting", "Hello Spring Transaction");
1027 return "home";
1028 }
1029 }
1030
1031 8)views/home.jsp
1032
```

```
1033     <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
1034     <%@ page session="false" %>
1035     <html>
1036     <head>
1037         <title>Home</title>
1038     </head>
1039     <body>
1040         <h1>${greeting}</h1>
1041     </body>
1042     </html>
1043
1044 9)project > right-click > Run As > Run on server
1045
1046 10)Database Table 생성
1047
1048     CREATE TABLE Card
1049     (
1050         consumerId    VARCHAR2(20),
1051         amount        NUMBER(2),
1052         CONSTRAINT card_consumerid_pk PRIMARY KEY(consumerId)
1053     );
1054
1055     CREATE TABLE Ticket
1056     (
1057         consumerId    VARCHAR2(20),
1058         countnum      NUMBER(2),
1059         CONSTRAINT ticket_consumerid_pk PRIMARY KEY(consumerId),
1060         CONSTRAINT ticket_consumerid_fk FOREIGN KEY(consumerId) REFERENCES
            Card(consumerId),
1061         CONSTRAINT ticket_countnum_ck CHECK(countnum < 5)
1062     );
1063
1064 11)src/main/java/com.example.vo.TicketVO.java
1065
1066     package com.example.vo;
1067
1068     public class TicketVO {
1069         private String consumerId;
1070         private String amount;
1071
1072         public String getConsumerId() {
1073             return consumerId;
1074         }
1075         public void setConsumerId(String consumerId) {
1076             this.consumerId = consumerId;
1077         }
1078         public String getAmount() {
1079             return amount;
1080         }
1081         public void setAmount(String amount) {
1082             this.amount = amount;
1083         }
1084     }
1085
```

```
1086 12)src/main/java/com.example.dao.TicketDao.java
1087
1088 package com.example.dao;
1089
1090 import org.slf4j.Logger;
1091 import org.slf4j.LoggerFactory;
1092 import org.springframework.beans.factory.annotation.Autowired;
1093 import org.springframework.jdbc.core.JdbcTemplate;
1094 import org.springframework.stereotype.Repository;
1095
1096 import com.example.vo.TicketVO;
1097
1098 @Repository("ticketDao")
1099 public class TicketDao {
1100     @Autowired
1101     private JdbcTemplate jdbcTemplate;
1102
1103     private static final Logger logger = LoggerFactory.getLogger(TicketDao.class);
1104
1105     public void buyTicket(final TicketVO ticketVO) {
1106         logger.warn("buyTicket()");
1107         logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
1108         logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
1109
1110         String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
1111         this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
1112
1113         sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
1114         this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
1115     }
1116 }
```

```
1118 13)views/buy_ticket.jsp
1119
1120 <%@ page language="java" contentType="text/html; charset=UTF-8"
1121     pageEncoding="UTF-8"%>
1122 <!DOCTYPE html>
1123 <html>
1124     <head>
1125         <meta charset="UTF-8">
1126         <title>Ticket 구매 창</title>
1127     </head>
1128     <body>
1129         <h1>카드 결제</h1>
1130
1131         <form action="buy_ticket_card" method="post">
1132             고객 아이디 : <input type="text" name="consumerId" > <br />
1133             티켓 구매수 : <input type="text" name="amount" > <br />
1134             <input type="submit" value="구매" >
1135         </form>
1136
1137     </body>
1138 </html>
1139
```

```
1140 14)views/buy_ticket_result.jsp
1141
1142 <%@ page language="java" contentType="text/html; charset=UTF-8"
1143     pageEncoding="UTF-8"%>
1144 <!DOCTYPE html>
1145 <html>
1146     <head>
1147         <meta charset="UTF-8">
1148         <title>Ticket 구매 결과 창</title>
1149     </head>
1150     <body>
1151         <h1>Ticket 구매 결과</h1>
1152         <ul>
1153             <li>고객 아이디 : ${ticketInfo.consumerId }</li>
1154             <li>구매 갯수 : ${ticketInfo.amount }</li>
1155         </ul>
1156     </body>
1157 </html>
1158
1159 15)HomeController.java 코드 추가
1160
1161 package com.javasoft.biz;
1162
1163 import org.springframework.beans.factory.annotation.Autowired;
1164 import org.springframework.stereotype.Controller;
1165 import org.springframework.ui.Model;
1166 import org.springframework.web.bind.annotation.RequestMapping;
1167 import org.springframework.web.bind.annotation.RequestMethod;
1168
1169 @Controller
1170 public class HomeController {
1171     @Autowired
1172     private TicketDao dao;
1173
1174     @RequestMapping(value = "/", method = RequestMethod.GET)
1175     public String home(Model model) {
1176         model.addAttribute("greeting", "Hello Spring Transaction");
1177         return "home";
1178     }
1179
1180     @RequestMapping("/buy_ticket")
1181     public String buy_ticket() {
1182         return "buy_ticket";
1183     }
1184
1185     @RequestMapping(value = "/buy_ticket_card", method = RequestMethod.POST)
1186     public String buy_ticket_card(TicketVO ticketVO, Model model) {
1187         logger.warn( "buy_ticket_card" );
1188         logger.warn( "고객 아이디 : " + ticketVO.getConsumerId() );
1189         logger.warn( "구매 갯수 : " + ticketVO.getAmount() );
1190
1191         dao.buyTicket(ticketVO);
1192
1193 }
```

```
1194         model.addAttribute("ticketInfo", ticketVO);
1195
1196         return "buy_ticket_result";
1197     }
1198 }
1199
1200 16)project > right-click > Run As > Run on server
1201
1202 17)http://localhost:8080/biz/buy\_ticket
1203     -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
1204     -하지만 구매 갯수를 5장 이상을 입력하면 오류 발생
1205         --Card table에는 구매 갯수가 5 이상의 값 입력이 가능하다.
1206         --하지만 Ticket table에는 ORA-02290: check constraint (SCOTT.TICKET_COUNTNUM_CHK) 위반
1207             이 발생했기 때문에 입력에 실패하게 된다.
1208     -이렇게 오류가 나오는 것이 정상이다.
1209     -그리고 치명적인 오류인 것은 Car 테이블에는 5장 이상 구매한 데이터는 입력이 되지만, Ticket table에는 체크 위반
1210     때문에 입력되지 않는다는 것이다.
1211     -현재 Card table에는 5장 이상이 입력된 상태이지만, Ticket table에는 에러가 발생했기 때문에 5장 이상이 입력되
1212     어 있지 않다.
1213     -두 테이블 모두 입력 성공하거나 입력 취소가 되어야 한다.
1214
1215 18)servlet-context.xml 에 코드 추가
1216
1217     <beans:bean name="transactionManager"
1218     class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
1219         <beans:property name="dataSource" ref="dataSource" />
1220     </beans:bean>
1221
1222     <tx:annotation-driven transaction-manager="transactionManager" />
1223
1224 19)TicketDao.java 코드 추가
1225
1226     @Transactional(propagation=Propagation.REQUIRED) <-- code 추가
1227     public void buyTicket(final TicketVO ticketVO) {
1228         logger.warn("buyTicket()");
1229         logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
1230         logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
1231
1232 20)project > right-click > Run As > Run on server
1233
1234 21)http://localhost:8080/biz/buy\_ticket
1235     -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
1236     -하지만 구매 갯수를 5장 이상을 입력하면 두 개의 테이블 모두 입력되지 않음.
1237     -다만 화면에 에러 메시지 display
1238     -transaction 성공했음.
```