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
    -In pom.xml > Dependencies tab > click [Add]
    -[Enter groupId, artifactId or sha1... 'spring jdbc' 입력
    -org.springframework spring-jdbc 선택
    -[OK]

```
   -pom.xml > right-click > Run As > Maven clean and Maven install

3)src/main/resource/oracle.properties 생성

   db.driver=oracle.jdbc.driver.OracleDriver
   db.url=jdbc:oracle:thin:@192.168.56.2:1521:ORCL
   db.username=scott
   db.password=tiger

4)pom.xml에 코드 추가

   <dependency>
      <groupId>com.oracle</groupId>
      <artifactId>ojdbc8</artifactId>
      <version>12.2</version>
   </dependency>

5)servlet-context.xml 코드 추가

   <context:component-scan base-package="com.example" />

   <context:property-placeholder location="classpath:oracle.properties" />
   <beans:bean id="dataSource"
      class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
      <beans:property name="driverClass" value="${db.driver}" />
      <beans:property name="url" value="${db.url}" />
      <beans:property name="username" value="${db.username}" />
      <beans:property name="password" value="${db.password}" />
   </beans:bean>

   <beans:bean name="template"
      class="org.springframework.jdbc.core.JdbcTemplate">
      <beans:property name="dataSource" ref="dataSource" />
   </beans:bean>

6)src/main/java/com.example.biz 이름변경
   -com.example.controller

7)com.example.controller.HomeController.java

   @Controller
   public class HomeController {

      private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

      @RequestMapping(value = "/", method = RequestMethod.GET)
      public String home(Model model) {
         model.addAttribute("greeting", "Hello Spring Transaction");
         return "home";
      }
   }

8)views/home.jsp

   <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
   <%@ page session="false" %>
   <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>
```

```
    </head>
    <body>
      <h1>Ticket 구매 결과</h1>
      <ul>
        <li>고객 아이디 : ${ticketInfo.consumerId }</li>
        <li>구매 갯수 : ${ticketInfo.amount }</li>
      </ul>
    </body>
  </html>
```

15)HomeController.java 코드 추가

```
  package com.javasoft.biz;

  import org.springframework.beans.factory.annotation.Autowired;
  import org.springframework.stereotype.Controller;
  import org.springframework.ui.Model;
  import org.springframework.web.bind.annotation.RequestMapping;
  import org.springframework.web.bind.annotation.RequestMethod;

  @Controller
  public class HomeController {
    @Autowired
    private TicketDao dao;

    @RequestMapping(value = "/", method = RequestMethod.GET)
    public String home(Model model) {
      model.addAttribute("greeting", "Hello Spring Transaction");
      return "home";
    }

    @RequestMapping("/buy_ticket")
    public String buy_ticket() {
      return "buy_ticket";

    }

    @RequestMapping(value = "/buy_ticket_card", method = RequestMethod.POST)
    public String buy_ticket_card(TicketVO ticketVO, Model model) {
      logger.warn( "buy_ticket_card" );
      logger.warn( "고객 아이디 : " + ticketVO.getConsumerId() );
      logger.warn( "구매 갯수 : " + ticketVO.getAmount() );

      dao.buyTicket(ticketVO);

      model.addAttribute("ticketInfo", ticketVO);

      return "buy_ticket_result";
    }
  }
```

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

17)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.

282 --하지만 Ticket table에는 ORA-02290: check constraint (SCOTT.TICKET_COUNTNUM_CK) 위반이 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 입력되어 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"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
294 <property name="dataSource" ref="dataSource" />
295 </bean>
296
297 2)JPA Transaction Manager 설정
298 -JPA를 사용할 경우
299
300 <bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
301 ...
302 </bean>
303
304 <bean name="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
305 <property name="entityManagerFactory" ref="entityManagerFactory" />
306 </bean>
307
308 3)Hibernate Transaction Manager 설정
309 -Hibernate를 사용하는 경우
310
311 <bean id="transactionManager"
class="org.springframework.orm.hibernate3.HibernateTransactionManager">
312 <property name="sessionFactory" ref="sessionFactory" />
313 </bean>
314
315 <bean id="sessionFactory"
class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
316 <property name="dataSource" ref="dataSource" />
317 ...
318 </bean>
319
320 4)Transaction 전파 속성
321 -org.springframework.transaction.TransactionDefinition interface
322 -2개 이상의 transaction이 작동할 때, 기존의 transaction에 참여하는 방법을 결정하는 속성.
323 -PROPAGATION_REQUIRED : 0
324 --method를 수행하는 데 transaction이 필요하다는 것을 의미
325 --현재 진행 중인 transaction이 존재하면, 해당 transaction을 사용한다.
326 --존재하지 않는다면 새로운 transaction을 생성한다.
327 --default, 즉 전체 처리한다.
328
329 -PROPAGATION_MANDATORY : 2
330 --method를 수행하는 데 transaction이 필요하다는 것을 의미한다.

```
331         --하지만 위의 REQUIRED와 달리, 진행중인 transaction이 존재하지 않을 경우 exception을 발생시킴
332         --transaction에 꼭 포함되어야 함
333
334       -PROPAGATION_REQUIRES_NEW : 3
335         --항상 새로운 transaction을 시작한다.
336         --기존 transaction이 존재하면 기존 transaction을 일시 중지하고 새로운 transaction을 시작한다.
337         --새로 시작된 transaction이 종료된 뒤에 기존 transaction이 계속된다.
338         --각자 transaction 처리(별도의 transaction 처리)
339
340       -PROPAGATION_SUPPORTS : 1
341         --method가 transaction을 필요로 하지 않지만, 기존 transaction이 존재할 경우 transaction을 사용한다는 것을 의
          미한다.
342         --진행중인 transaction이 존재하지 않더라고 method는 정상적으로 동작한다.
343         --기존 transaction에 의존
344
345       -PROPAGATION_NOT_SUPPORTED : 4
346         --method가 transaction을 필요로 하지 않음을 의미한다.
347         --SUPPORTS와 달리 진행 중인 transaction이 존재할 경우 method가 실행되는 동안 transaction은 일시 중지되며,
          method 실행이 종료된 뒤에 transaction을 계속 진행한다.
348         --transaction에 포함하지 않음 -> transaction이 없는 것과 동일
349
350       -PROPAGATION_NEVER : 5
351         --method가 transaction을 필요로 하지 않으며, 만약 진행 중인 transaction이 존재하면 exception을 발생시킴.
352         --transaction에 절대 포함하지 않음.
353
354       -PROPAGATION_NESTED : 6
355         --기존 transaction이 존재하면, 기존 transaction에 중첩된 transaction에서 method를 실행한다.
356         --기존 transaction이 존재하지 않으면 REQUIRED와 동일하게 동작한다.
357         --이 기능은 JDBC 3.0 driver를 사용할 때에만 적용된다.
358
359       REQUIRED <----> REQUIRES_NEW
360       MANDATORY <---> NEVER
361       SUPPORTS <---> NOT_SUPPORTED
362
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"
          class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
384         <beans:property name="dataSource" ref="dataSource" />
```

```
385       </beans:bean>
386
387     2)TicketDao.java 코드 추가
388
389       @Autowired
390       private PlatformTransactionManager transactionManager;  <-- 코드 추가
391
392       public void buyTicket(final TicketVO ticketVO) {
393         logger.warn("buyTicket()");
394         logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
395         logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
396
397         TransactionDefinition definition = new DefaultTransactionDefinition();                  <--코드 추
           가
398         TransactionStatus status = this.transactionManager.getTransaction(definition);         <--코드 추
           가
399
400         try {                                                                                    <--코드
           추가
401           String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
402           this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
403
404           sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
405           this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
406           this.transactionManager.commit(status);                                              <--코드
             추가
407         }catch(Exception e) {                                                                    <--코드
           추가
408           e.printStackTrace();                                                                 <--코드
             추가
409           this.transactionManager.rollback(status);                                            <--코드
             추가
410         }                                                                                        <--코드
           추가
411       }
412
413     3)project > right-click > Run As > Run on server
414
415     4)http://localhost:8080/biz/buy_ticket
416       -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
417       -하지만 구매 갯수를 5장 이상을 입력하면 두 개의 테이블 모두 입력되지 않음.
418       -transaction 성공했음.
419
420
421   5. TransactionTemplate 이용한 Transaction 처리하기
422     1)기본적으로 사용한 PlatformTransactionManager interface보다 더 많이 사용되는 객체는 TransactionTemplate 이
      다.
423     2)Transaction을 처리하기 위해 PlatformTransactionManager의 메소드를 직접 사용해도 되지만 try/catch 블록을 써야
      하는 번거로움이 발생한다.
424     3)Transaction 안에서 작업 중에 예외가 발생한 경우에는 Transaction을 롤백해주도록 만들어야 하기 때문이다.
425     4)그래서 PlatformTransactionManager의 메소드를 직접 사용하는 대신 Template/Callback 방식의
      TransactionTemplate을 이용하면 편리하다.
426
427
428     5)applicationContext.xml 코드 추가
429
430       <beans:bean name="transactionTemplate"
```

```
    class="org.springframework.transaction.support.TransactionTemplate">
431   <beans:property name="transactionManager" ref="transactionManager" />
432 </beans:bean>
433
434 6)TicketDao.java 코드 수정 및 추가
435
436 @Repository("ticketDao")
437 public class TicketDao {
438   @Autowired
439   private JdbcTemplate jdbcTemplate;
440
441   @Autowired
442   private TransactionTemplate transactionTemplate;
443
444
445   public void buyTicket(final TicketVO ticketVO) {
446     logger.warn("buyTicket()");
447     logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
448     logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
449
450     this.transactionTemplate.execute(new TransactionCallbackWithoutResult() {
451
452       @Override
453       protected void doInTransactionWithoutResult(TransactionStatus status) {
454         String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
455         TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
          ticketVO.getAmount());
456
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 }
468
469 4)project > right-click > Run As > Run on server
470
471 5)http://localhost:8080/biz/buy_ticket
472   -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
473   -하지만 구매 갯수를 5장 이상을 입력하면 두 개의 테이블 모두 입력되지 않음.
474   -다만 화면에 에러 메시지 display
475   -transaction 성공했음.
476
477
478 6. Transaction 전파에 관한 Lab
479   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     }
488
489   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     }
527
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(),
                  ticketVO.getAmount());
589 
590               sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
591               TicketDao.this.jdbcTemplate.update(sql, ticketVO.getConsumerId(),
                  ticketVO.getAmount());
592             }
593 
594           });
595         }
```

```
596     }
597
598   5)servlet-context.xml
599
600     ...
601     <beans:bean name="transactionTemplate1"
        class="org.springframework.transaction.support.TransactionTemplate">
602       <beans:property name="transactionManager" ref="transactionManager" />
603       <beans:property name="propagationBehavior" value="0" />
604     </beans:bean>
605     <beans:bean name="transactionTemplate2"
        class="org.springframework.transaction.support.TransactionTemplate">
606       <beans:property name="transactionManager" ref="transactionManager" />
607       <beans:property name="propagationBehavior" value="0" />
608     </beans:bean>
609     <beans:bean name="ticketDao" class="com.example.dao.TicketDao" >
610       <beans:property name="jdbcTemplate" ref="template" />
611       <beans:property name="transactionTemplate1" ref="transactionTemplate1" />
612     </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     -@Transaction 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> tag
      를 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_UNCOMMITED, READ_COMMITED, 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 샐제로 transaction을 적용하는 것은 아니다.
644     -실제로 transaction을 적용하는 것은 AOP를 통해서 이루어진다.
645
```

4)Annotation 기반 Transaction 설정
-@Transactional annotation을 사용해서 transaction 설정 가능.
-@Transactional annotation 의 주요 속성
--propagation : transaction 전파 규칙 설정.
---org.springframework.transaction.annotation.Propagation 나열형으로 정의돼 있음.
---기본값은 Propagation.REQUIRED이다.
--isolation : transaction 격리 level 설정.
---org.springframework.transaction.annotation.Isolation 나열형으로 정의돼 있음.
--readOnly : 읽기 전용 여부 설정. boolean 값을 설정하며, 기본값은 false이다.
--rollbackFor : transaction을 rollback할 Exception Type을 설정.
---예 : rollbackFor={Exception.class}
--noRollbackFor : transaction을 rollback하지 않을 Exception Type 설정.
---예 : noRollbackFor={ItemNotFoundException.class}
--timeout : transaction의 timeout 시간을 초 단위로 지정.
-@Transactional annotation이 적용될 Spring Bean에 Transaction을 실제로 적용하려면 다음과 같이 <tx:annotation-driven>을 설정해야 한다.
<tx:annotation-driven transaction-manager="transactionManager" />
-<tx:annotation-driven> tag 속성
--transaction-manager : 사용할 PlatformTransaction Manager bean 이름.
--기본값은 transactionManager
--proxy-target-class : class에 대해서 proxy를 생성 여부.
---true는 CGLIB를 이용해서 proxy를 생성
---false는 Java dynamic proxy를 이용해서 proxy를 생성, 기본값
--order : Advice 적용 순서 : int의 최대값(가장 낮은 순위).

8. Lab : tx namespace를 이용한 Transaction 설정
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
-In pom.xml > Dependencies tab > click [Add]
-[Enter groupId, artifactId or sha1... 'spring jdbc' 입력
-org.springframework spring-jdbc 선택
-[OK]
-pom.xml > right-click > Run As > Maven clean and Maven install

3)src/main/resource/oracle.properties 생성

db.driver=oracle.jdbc.driver.OracleDriver
db.url=jdbc:oracle:thin:@192.168.56.2:1521:ORCL
db.username=scott
db.password=tiger

4)pom.xml에 코드 추가

<dependency>
<groupId>com.oracle</groupId>
<artifactId>ojdbc8</artifactId>
<version>12.2</version>
</dependency>

5)servlet-context.xml 코드 추가

```
    <context:component-scan base-package="com.example" />

    <context:property-placeholder location="classpath:oracle.properties" />
    <beans:bean id="dataSource"
      class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
      <beans:property name="driverClass" value="${db.driver}" />
      <beans:property name="url" value="${db.url}" />
      <beans:property name="username" value="${db.username}" />
      <beans:property name="password" value="${db.password}" />
    </beans:bean>

    <beans:bean name="template"
      class="org.springframework.jdbc.core.JdbcTemplate">
      <beans:property name="dataSource" ref="dataSource" />
    </beans:bean>

6)src/main/java/com.example.biz 이름변경
  -com.example.controller

7)com.example.controller.HomeController.java

    @Controller
    public class HomeController {

      private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

      @RequestMapping(value = "/", method = RequestMethod.GET)
      public String home(Model model) {
        model.addAttribute("greeting", "Hello Spring Transaction");
        return "home";
      }
    }

8)views/home.jsp

    <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
    <%@ page session="false" %>
    <html>
    <head>
      <title>Home</title>
    </head>
    <body>
      <h1>${greeting}</h1>
    </body>
    </html>

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

10)Database Table 생성

      CREATE TABLE Card
      (
        consumerId    VARCHAR2(20),
        amount        NUMBER(2),
        CONSTRAINT card_consumerid_pk PRIMARY KEY(consumerId)
      );

```

```
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(?, ?)";
```

```
        this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());

        sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
        this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
      }
    }
```

13)views/buy_ticket.jsp

```
    <%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
    <!DOCTYPE html>
    <html>
      <head>
        <meta charset="UTF-8">
        <title>Ticket 구매 창</title>
      </head>
      <body>
        <h1>카드 결제</h1>

        <form action="buy_ticket_card" method="post">
          고객 아이디 : <input type="text" name="consumerId" > <br />
          티켓 구매수 : <input type="text" name="amount" > <br />
          <input type="submit" value="구매" >
        </form>

      </body>
    </html>
```

14)views/buy_ticket_result.jsp

```
    <%@ page language="java" contentType="text/html; charset=UTF-8"
       pageEncoding="UTF-8"%>
    <!DOCTYPE html>
    <html>
      <head>
        <meta charset="UTF-8">
        <title>Ticket 구매 결과 창</title>
      </head>
      <body>
        <h1>Ticket 구매 결과</h1>
        <ul>
          <li>고객 아이디 : ${ticketInfo.consumerId }</li>
          <li>구매 갯수 : ${ticketInfo.amount }</li>
        </ul>
      </body>
    </html>
```

15)HomeController.java 코드 추가

```
    package com.javasoft.biz;

    import org.springframework.beans.factory.annotation.Autowired;
    import org.springframework.stereotype.Controller;
    import org.springframework.ui.Model;
    import org.springframework.web.bind.annotation.RequestMapping;
    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
890       @RequestMapping(value = "/buy_ticket_card", method = RequestMethod.POST)
891       public String buy_ticket_card(TicketVO ticketVO, Model model) {
892         logger.warn( "buy_ticket_card" );
893         logger.warn( "고객 아이디 : " + ticketVO.getConsumerId() );
894         logger.warn( "구매 갯수 : " + ticketVO.getAmount() );
895
896         dao.buyTicket(ticketVO);
897
898         model.addAttribute("ticketInfo", ticketVO);
899
900         return "buy_ticket_result";
901       }
902     }
903
904   16)project > right-click > Run As > Run on server
905
906   17)http://localhost:8080/biz/buy_ticket
907     -구매 갯수를 5이하로 입력하면 정상적으로 처리됨.
908     -하지만 구매 갯수를 5장 이상을 입력하면 오류 발생
909       --Card table에는 구매 갯수가 5 이상의 값 입력이 가능하다.
910       --하지만 Ticket table에는 ORA-02290: check constraint (SCOTT.TICKET_COUNTNUM_CK) 위반이 발생
          했기 때문에 입력에 실패하게 된다.
911     -이렇게 오류가 나오는 것이 정상이다.
912     -그리고 치명적인 오류인 것은 Car 테이블에는 5장 이상 구매한 데이터는 입력이 되지만, Ticket table에는 체크 위반 때문에
        입력되지 않는다는 것이다.
913     -현재 Card table에는 5장 이상이 입력된 상태이지만, Ticket table에는 에러가 발생했기 때문에 5장 이상이 입력되어 있지
        않다.
914     -두 테이블 모두 입력 성공하거나 입력 취소가 되어야 한다.
915
916   18)AspectJ Weaver library 설치
917     -Maven Repository에서 'aspectj weaver'으로 검색
918     -aspectj weaver 1.9.2 버전을 pom.xml에 추가
919
920       <dependency>
921         <groupId>org.aspectj</groupId>
922         <artifactId>aspectjweaver</artifactId>
923         <version>1.9.2</version>
924       </dependency>
925
```

19)Spring AOP library 설치
-Maven Repository에서 'spring aop'으로 검색
-aspectj weaver 4.3.20 버전을 pom.xml에 추가

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-aop</artifactId>
  <version>4.3.20.RELEASE</version>
</dependency>
```

-Maven Install

20)servlet-context.xml에서
-aop, tx check
-applicationContext.xml에 다음의 코드를 추가한다.

```
<beans:bean name="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <beans:property name="dataSource" ref="dataSource" />
</beans:bean>

<tx:advice id="txAdvice" transaction-manager="transactionManager">
  <tx:attributes>
    <tx:method name="buyTicket" propagation="REQUIRED"/>
  </tx:attributes>
</tx:advice>

<aop:config>
  <aop:pointcut expression="execution(public * com.example.dao.TicketDao.*(..))"
  id="servicePublicMethod"/>
  <aop:advisor advice-ref="txAdvice" pointcut-ref="servicePublicMethod"/>
</aop:config>
```

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

22)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.

9. Lab : Annotation 기반 Transaction 설정
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
-In pom.xml > Dependencies tab > click [Add]
-[Enter groupId, artifactId or sha1... 'spring jdbc' 입력
-org.springframework spring-jdbc 선택
-[OK]
-pom.xml > right-click > Run As > Maven clean and Maven install

3)src/main/resource/oracle.properties 생성

```
db.driver=oracle.jdbc.driver.OracleDriver
db.url=jdbc:oracle:thin:@192.168.56.2:1521:ORCL
db.username=scott
db.password=tiger
```

4)pom.xml에 코드 추가

```
<dependency>
  <groupId>com.oracle</groupId>
  <artifactId>ojdbc8</artifactId>
  <version>12.2</version>
</dependency>
```

5)servlet-context.xml 코드 추가

```
<context:component-scan base-package="com.example" />

<context:property-placeholder location="classpath:oracle.properties" />
<beans:bean id="dataSource"
  class="org.springframework.jdbc.datasource.SimpleDriverDataSource">
  <beans:property name="driverClass" value="${db.driver}" />
  <beans:property name="url" value="${db.url}" />
  <beans:property name="username" value="${db.username}" />
  <beans:property name="password" value="${db.password}" />
</beans:bean>

<beans:bean name="template"
  class="org.springframework.jdbc.core.JdbcTemplate">
  <beans:property name="dataSource" ref="dataSource" />
</beans:bean>
```

6)src/main/java/com.example.biz 이름변경
-com.example.controller

7)com.example.controller.HomeController.java

```
@Controller
public class HomeController {

  private static final Logger logger = LoggerFactory.getLogger(HomeController.class);

  @RequestMapping(value = "/", method = RequestMethod.GET)
  public String home(Model model) {
    model.addAttribute("greeting", "Hello Spring Transaction");
    return "home";
  }
}
```

8)views/home.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ page session="false" %>
<html>
<head>
  <title>Home</title>
```

```
  </head>
  <body>
    <h1>${greeting}</h1>
  </body>
  </html>

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

10)Database Table 생성

    CREATE TABLE Card
    (
      consumerId    VARCHAR2(20),
      amount       NUMBER(2),
      CONSTRAINT card_consumerid_pk PRIMARY KEY(consumerId)
    );

    CREATE TABLE Ticket
    (
      consumerId    VARCHAR2(20),
      countnum       NUMBER(2),
      CONSTRAINT ticket_consumerid_pk PRIMARY KEY(consumerId),
      CONSTRAINT ticket_consumerid_fk FOREIGN KEY(consumerId) REFERENCES
      Card(consumerId),
      CONSTRAINT ticket_countnum_ck CHECK(countnum < 5)
    );

11)src/main/java/com.example.vo.TicketVO.java

  package com.example.vo;

  public class TicketVO {
    private String consumerId;
    private String amount;

    public String getConsumerId() {
      return consumerId;
    }
    public void setConsumerId(String consumerId) {
      this.consumerId = consumerId;
    }
    public String getAmount() {
      return amount;
    }
    public void setAmount(String amount) {
      this.amount = amount;
    }
  }

12)src/main/java/com.example.dao.TicketDao.java

  package com.example.dao;

  import org.slf4j.Logger;
  import org.slf4j.LoggerFactory;
  import org.springframework.beans.factory.annotation.Autowired;
  import org.springframework.jdbc.core.JdbcTemplate;
```

```
import org.springframework.stereotype.Repository;

import com.example.vo.TicketVO;

@Repository("ticketDao")
public class TicketDao {
  @Autowired
  private JdbcTemplate jdbcTemplate;

  private static final Logger logger = LoggerFactory.getLogger(TicketDao.class);

  public void buyTicket(final TicketVO ticketVO) {
    logger.warn("buyTicket()");
    logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
    logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());

    String sql = "INSERT INTO card(consumerid, amount) VALUES(?, ?)";
    this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());

    sql = "INSERT INTO ticket(consumerid, countnum) VALUES(?, ?)";
    this.jdbcTemplate.update(sql, ticketVO.getConsumerId(), ticketVO.getAmount());
  }
}
```

13)views/buy_ticket.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Ticket 구매 창</title>
  </head>
  <body>
    <h1>카드 결제</h1>

    <form action="buy_ticket_card" method="post">
      고객 아이디 : <input type="text" name="consumerId" > <br />
      티켓 구매수 : <input type="text" name="amount" > <br />
      <input type="submit" value="구매" >
    </form>

  </body>
</html>
```

14)views/buy_ticket_result.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
   pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Ticket 구매 결과 창</title>
  </head>
  <body>
```

```
        <h1>Ticket 구매 결과</h1>
        <ul>
          <li>고객 아이디 : ${ticketInfo.consumerId }</li>
          <li>구매 갯수 : ${ticketInfo.amount }</li>
        </ul>
      </body>
    </html>
```

15)HomeController.java 코드 추가

```
package com.javasoft.biz;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

@Controller
public class HomeController {
  @Autowired
  private TicketDao dao;

  @RequestMapping(value = "/", method = RequestMethod.GET)
  public String home(Model model) {
    model.addAttribute("greeting", "Hello Spring Transaction");
    return "home";
  }

  @RequestMapping("/buy_ticket")
  public String buy_ticket() {
    return "buy_ticket";

  }

  @RequestMapping(value = "/buy_ticket_card", method = RequestMethod.POST)
  public String buy_ticket_card(TicketVO ticketVO, Model model) {
    logger.warn( "buy_ticket_card" );
    logger.warn( "고객 아이디 : " + ticketVO.getConsumerId() );
    logger.warn( "구매 갯수 : " + ticketVO.getAmount() );

    dao.buyTicket(ticketVO);

    model.addAttribute("ticketInfo", ticketVO);

    return "buy_ticket_result";
  }
}
```

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

17)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_CK) 위반이 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)servlet-context.xml 에 코드 추가

```
<beans:bean name="transactionManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
  <beans:property name="dataSource" ref="dataSource" />
</beans:bean>

<tx:annotation-driven transaction-manager="transactionManager" />
```

19)TicketDao.java 코드 추가

```
@Transactional(propagation=Propagation.REQUIRED)   <-- code 추가
public void buyTicket(final TicketVO ticketVO) {
  logger.warn("buyTicket()");
  logger.warn("ticketVO.getConsumerId() : " + ticketVO.getConsumerId());
  logger.warn("ticketVO.getAmount() : " + ticketVO.getAmount());
```

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

21)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.