제 15 강 JPA

우리가 사용하는 대부분 프로그램은 사용자가 입력한 데이터나 비즈니스 로직 수행 결과로 얻은 데이터를 재사용할 수 있도록 데이터베이스에 저장한다. 하지만 자바 객체와 데이터베이스 테이블이 정확하게 일치하는지는 않는다. 따라서 둘 사이를 매핑하기 위한 많은 SQL 구문과 자바 코드가 필요하다.

ORM은 이렇게 정확하게 일치하지 않는 자바 객체와 테이블 사이를 매핑 해 준다. 다시 말하면 ORM은 자바객체에 저장된 데이터를 테이블의 Row 정보로 저장하고 반대로 테이블에 저장된 Row 정보를 자바 객체로 매핑 해 준다. 이 과정에서 사용되는 SQL 구문과 자바 코드는 ORM 프레임워크가 자동으로 생성해 준다.

우리는 지금까지 스프링 JDBC나 MyBatis를 이용하여 자바 객체와 테이블을 매핑 해왔다. 하지만 어떤 DB연동기술을 이용하더라도 SQL 명령어를 자바 클래스나 외부 XML파일에 작성해야 했다. 이렇게 작성된 SQL문은 유지보수과정에서 지속적으로 수정되며 새로운 SQL문이 추가되기도 한다.

ORM 프레임워크의 가장 큰 특징은 DB 연동에 필요한 SQL문을 자동으로 생성한다는 것이다. 이렇게 생성되는 SQL문은 DBMS가 변경될 때 자동으로 변경된다. 다만 ORM 환경설정 파일 어딘가에 DBMS가 변경되었다는 것만 수정해 주면 된다.

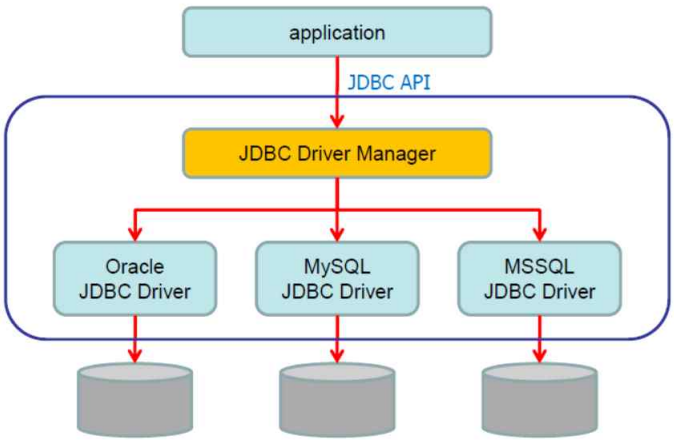
사실 개발자들은 오래전부터 객체와 테이블간의 불일치 때문에 발생하는 여러 문제점을 해결하기 위해 다양한 노력을 기울여 왔다. 그 과정에서 등장한 것이 EJB의 EntityBean 기술이다. 그 후 Hibernate가 등장하면선 기능도 추가되고 성능도 향상되어 ORM 프레임 워크의 대표가 되었다.

또 Hibernate 외에도 TopLink나 Codebase 같은 다른 ORM 프레임워크들도 등장하기 시작했고 이러한 프레임워크들에 대한 표준화 작업이 필요해 졌다.

ORM 프레임워크에 대한 표준화 작업의 결과가 JPA인 것이다.

1, JPA의 특징

▢ JPA(Java Persistence API)는 모든 ORM 구현체(ORM 프레임워크)들의 공통 인터페이스를 제공한다. 그럼 JPA와 JDBC를 비교하면서 살펴보자.

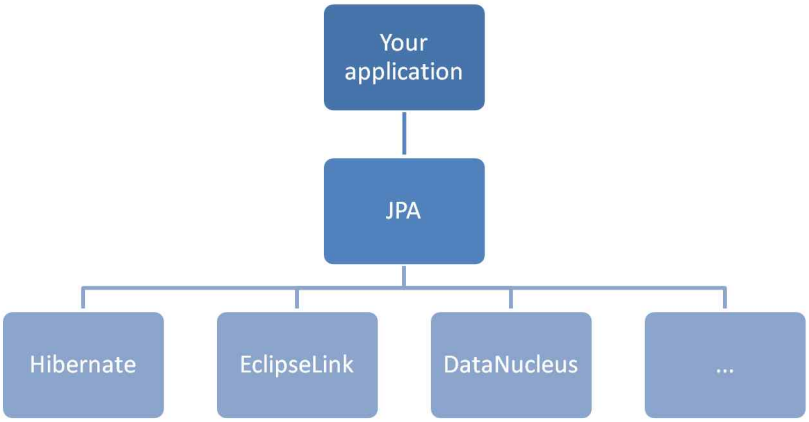


JDBC 프로그램 구조

▢ JDBC는 특정 DBMS에 종속되지 않는 데이터베이스 연동 구현을 지원한다. DB 연동 로직을 구현할 때 JDBC API(java.sql)의 인터페이스들을 이용하면 실질적인 데이터베이스 연동 처리는 해당 DBMS의 드라이버 클래스들이 담당하는 구조이다.

▢ 따라서 DBMS가 변경되는 상황에서도 드라이버만 변경하면 JDBC API를 이용하는 애플리케이션은 수정하지 않는다. 즉 개발당시에는 mySQL을 사용하다가 실제 서비스가 오픈될 때는 Oracle로 변경할 수도 있다는 것이다.

▢ JPA도 JDBC와 마찬가지 원리로 동작한다. 애플리케이션을 구현할 때 JPA API(javax.persistence)를 이용하면 개발 당시에는 Hibernate를 ORM 프레임워크로 사용하다가 실제 서비스가 시작될때는 TopLink로 변경할 수 있다.



JPA 프로그램 구조

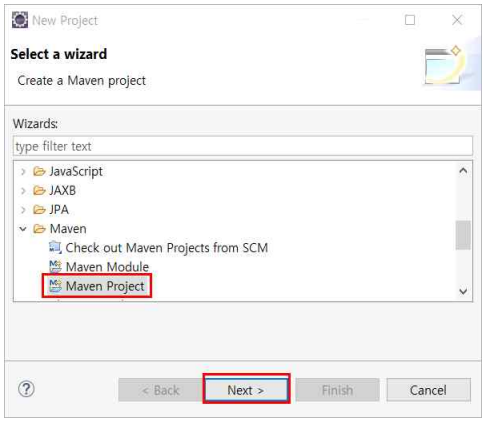
2. 실습 : Maven - JPA 프로젝트

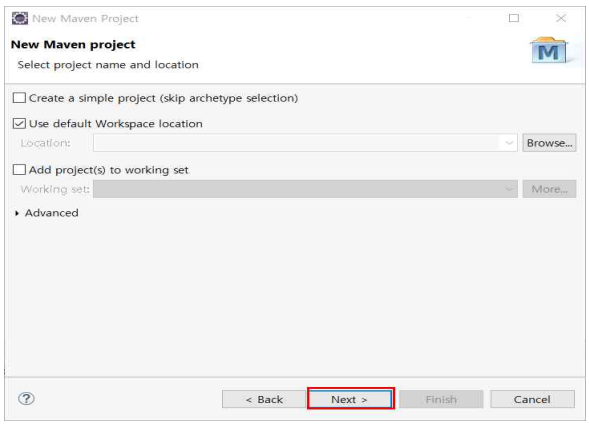
① JPA 프로젝트 생성

▢ JPA 프로젝트는 이클립스의 JPA Project 마법사를 이용해도 되지만 JPA Project는 추가로 설정해야 하는 작업들이 있어서 우리는 익숙한 Maven 기반의 프로젝트를 생성하도록 하겠다.

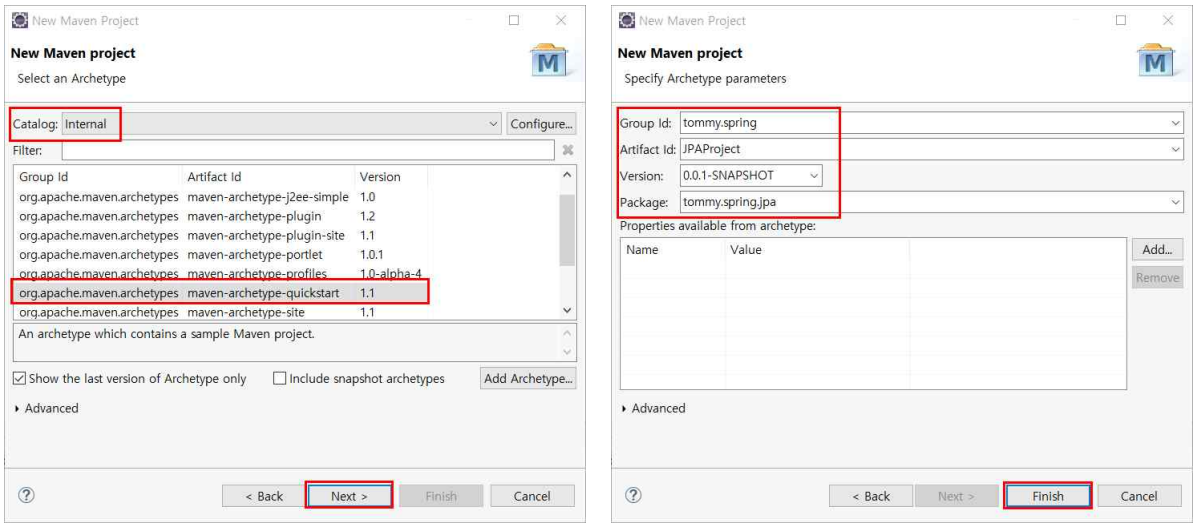
▢ 이클립스에서 [File]-[New] 메뉴에서 [Maven Project]를 선택한다.

▢ 다음 화면에서는 변경할 내용이 없으므로 그냥 [Next]로 넘어간다.





▢ 우선 자바 프로젝트를 생성할 것이다. 아래와 같이 프로젝트를 생성하도록 하자.



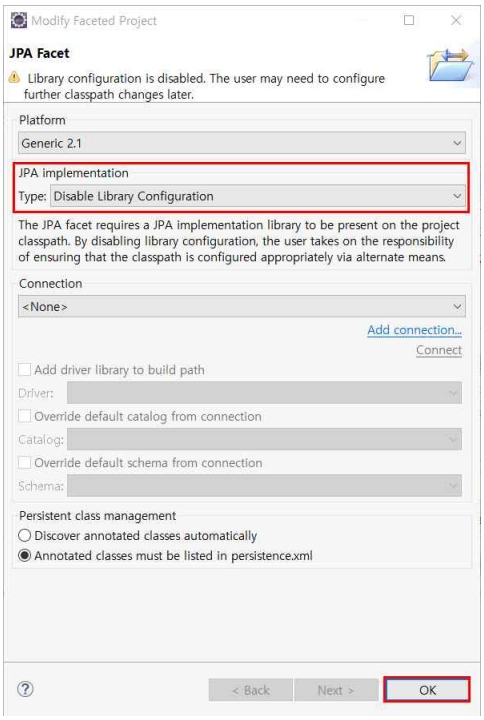
▢ 프로젝트가 완성된 후 보면 JRE System Library 버전이 맞지 않을 것이다. 이를 수정하자.

▢ [JPAProject]에서 마우스 우 클릭 후 [Properties]를 선택 왼쪽의 [Project Facets]를 선택한 후 오른쪽에 밑줄이 되어 있는 'Convert to Faceted from' 링크를 클릭한다.

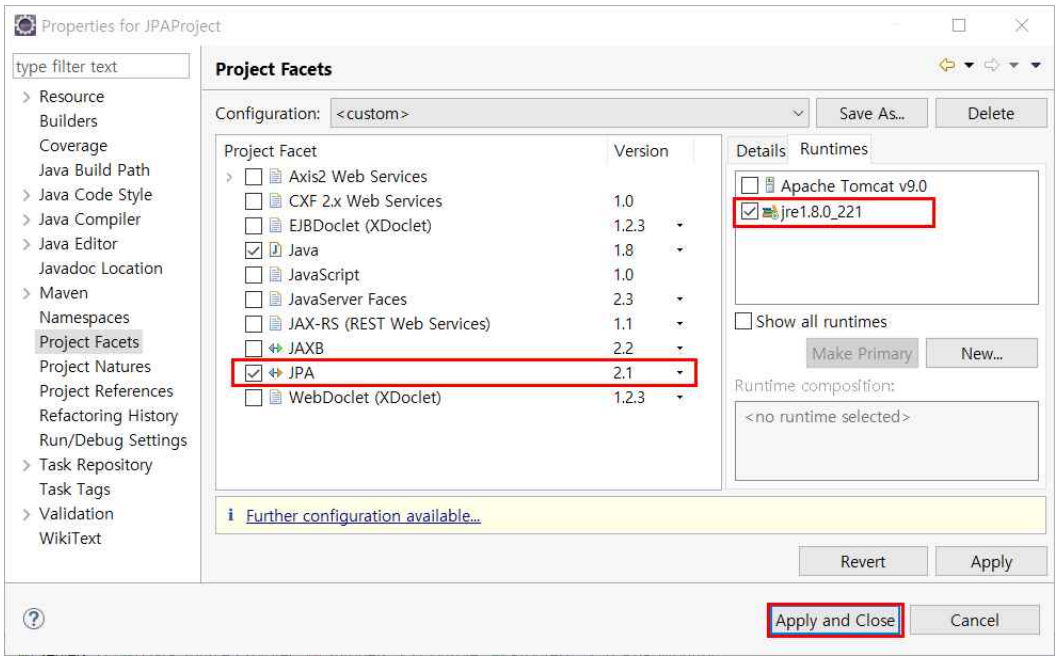
▢ 그리고 아래와 같이 'JPA' 항목을 체크하고 JDK 버전을 변경한다.

▢ 이때 만약 'Further configuration required...' 메시지가 출력된다면 우선 해당 링크를 클릭한다.

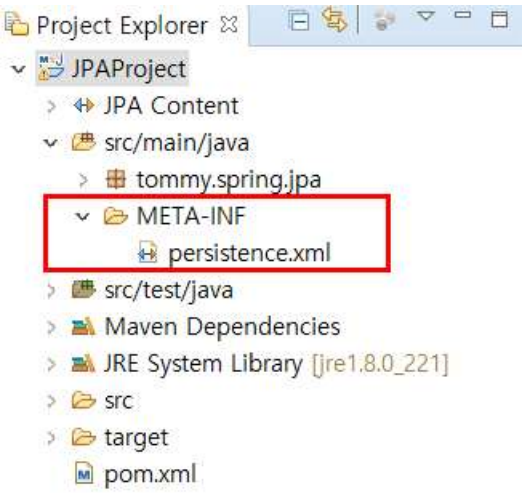
▢ 위 링크를 클리한 후 아래와 같이 설정하고 OK를 클릭하자.



▢ 이제 아래와 같이 JRE 버전을 수정하면 된다.



▢ 아래와 같이 src/main/java 폴더에 META-INF 폴더와 그 아래에 JPA 환경설정 파일인 persistence.xml 파일이 생성되었다면 성공적으로 프로젝트가 생성된 것이다.



② JPA 라이브러리 내려받기

▢ 프로젝트에 JPA 관련 라이브러리들을 추가하기 위해서 pom.xml 파일을 수정하자.

```
<!-- 상단 부분 생략 -->
<dependencies>
        <!-- Oracle Driver -->
        <dependency>
                <groupId>com.oracle</groupId>
                <artifactId>ojdbc8</artifactId>
                <version>18.3</version>
        </dependency>
        <!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-entitymanager -->
        <dependency>
```

```
10                      <groupId>org.hibernate</groupId>
11                      <artifactId>hibernate-entitymanager</artifactId>
12                      <version>5.4.18.Final</version>
13              </dependency>
14              <dependency>
15                      <groupId>junit</groupId>
16                      <artifactId>junit</artifactId>
17                      <version>4.12</version>
18                      <scope>test</scope>
19              </dependency>
20      </dependencies>
21      <build>
22              <plugins>
23                      <plugin>
24                              <groupId>org.apache.maven.plugins</groupId>
25                              <artifactId>maven-compiler-plugin</artifactId>
26                              <version>3.6.2</version>
27                              <configuration>
28                                      <source>1.8</source>
29                                      <target>1.8</target>
30                                      <encoding>UTF-8</encoding>
31                              </configuration>
32                      </plugin>
33              </plugins>
34      </build>
        <!-- 하단 부분 생략 -->
```

③ 엔티티 클래스 매핑

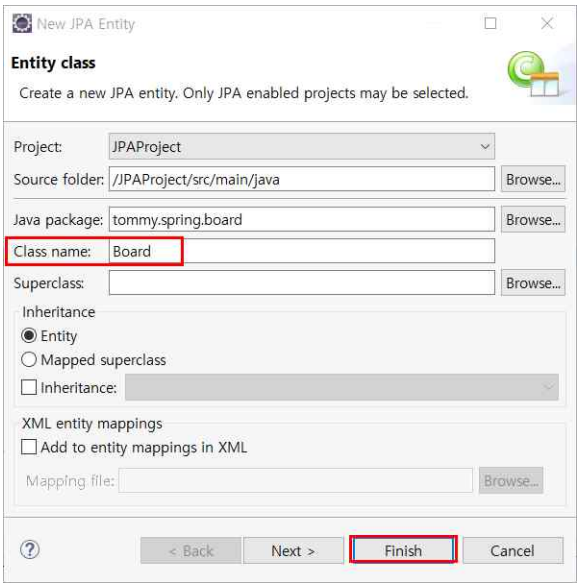
☐ 우선 데이터베이스 테이블과 매핑될 영속 클래스를 작성하고 매핑관련 설정을 추가해야 한다. 엔티티 클래스를 작성할 때 특별한 제약조건이나 규칙이 있는 것은 아니므로 일반적인 VO 클래스를 만들 때처럼 작성하면 된다. 하지만 될 수 있으면 이클립스에서 제공하는 JPA Entity 생성 기능을 이용하는 것이 좋다.

☐ src/main/java 폴더와 src/main/test 폴더 안에 있는 tommy.spring.jpa 패키지를 삭제한다.

☐ src/main/java 소스 폴더를 클릭하여 tommy.spring.board 패키지를 만든다.

☐ 작성한 패키지를 클릭하여 [New]-[JPA Entity]를 선택한다.

▢ ‘Class Name’에 ‘Board’라고 입력한다.



▢ 이 때 /META-INF/persistence.xml 파일을 열어보면 아래와 같이 Board라는 엔티티 클래스가 자동으로 등록된 것을 확인할 수 있다.

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
6           <persistence-unit name="JPAProject">
7                 <class>tommy.spring.board.Board</class>
8           </persistence-unit>
9  </persistence>
```

▢ 이제 엔티티 클래스에 JPA 매핑관련 어노테이션을 설정한다. 그리고 엔티티 클래스의 모든 멤버변수를 private으로 선언한다. 일반적인 프로그램에서는 객체를 식별하기 위해서 유일 식별자를 사용하지 않지만 영속 객체가 테이블에 매핑 될 때 객체 식별 방법이 필요하므로 유일 식별자를 소유하는 클래스로 작성해야 한다.

■ /src/main/java/Board.java

```
1  package tommy.spring.board;

2  import java.util.Date;

3  import javax.persistence.Entity;
4  import javax.persistence.GeneratedValue;
5  import javax.persistence.Id;
6  import javax.persistence.Table;
7  import javax.persistence.Temporal;
8  import javax.persistence.TemporalType;
```

```
 9  @Entity
10  @Table(name = "MYBOARD")
11  public class Board {

12          @Id
13          @GeneratedValue
14          private int seq;
15          private String title;
16          private String writer;
17          private String content;
18          @Temporal(TemporalType.DATE)
19          private Date regDate = new Date();
20          private int cnt;

            // getter, setter 추가
            // Generate toString() 추가

    }
```

■ 참고 : Board 엔티티 클래스에 설정된 어노테이션

어노테이션	의미
@Entity	@Entity가 설정된 클래스를 엔티티 클래스라고 하며 @Entity가 붙은 클래스는 테이블과 매핑된다.
@Table	엔티티와 관련된 테이블을 매핑한다. name 속성을 사용하여 BOARD 테이블과 매핑한다. 생략하면 클래스 이름이 테이블 이름과 매핑된다.
@Id	엔티티 클래스의 필수 어노테이션으로서 특정 변수를 테이블의 기본 키와 매핑한다. 예제에서는 seq변수를 테이블의 SEQ 컬럼과 매핑한다. @Id가 없는 엔티티 클래스는 JPA가 처리하지 못한다.
@GeneratedValue	@Id가 선언된 필드에 기본 키를 자동으로 생성하여 할당할 때 사용한다. 다양한 옵션이 있지만 @GeneratedValue만 사용하면 데이터베이스에 따라서 자동으로 결정된다. Oracle은 시퀀스를 이용하여 처리한다.
@Temporal	날짜 타입의 변수에 선언하여 날짜 타입을 매핑할 때 사용한다. TemporalType의 DATE, TIME, TIMESTAMP중 하나를 선택할 수 있다.

④ persistence.xml 파일 작성

☐ JPA는 persistence.xml 파일을 사용하여 필요한 설정 정보를 관리한다. 이 설정 파일이 META-INF 폴더 아래에 있으면 별도의 설정없이 JPA가 인식한다.

```
    <!-- 상단 부분 생략 -->
1   <persistence-unit name="JPAProject">
2           <class>tommy.spring.board.Board</class>
3           <properties>
                    <!-- 필수 속성 -->
4                   <property name="javax.persistence.jdbc.driver"
```

```
                                                value="oracle.jdbc.driver.OracleDriver"/>
5                   <property name="javax.persistence.jdbc.user" value="mytest"/>
6                   <property name="javax.persistence.jdbc.password" value="mytest"/>
7                   <property name="javax.persistence.jdbc.url"
                            value="jdbc:oracle:thin:@localhost:1521/XEPDB1"/>
8                   <property name="hibernate.dialect"
                                    value="org.hibernate.dialect.Oracle9Dialect"/>
                    <!-- 옵션 -->
9                   <property name="hibernate.show_sql" value="true"/>
10                  <property name="hibernate.format_sql" value="true"/>
11                  <property name="hibernate.use_sql_comments" value="false"/>
12                  <property name="hibernate.id.new_generator_mappings" value="true"/>
13                  <property name="hibernate.hbm2ddl.auto" value="create"/>
14          </properties>
15  </persistence-unit>
    <!-- 하단 부분 생략 -->
```

▢ persistence.xml 파일은 JPA에서 메인 환경설정 파일이다.
persistence.xml 파일에는 <persistence>를 루트 엘리먼트로 사용하며 영속성 유닛(</persistence-unit>)이 설정 되어 있다. 영속성 유닛은 연동할 데이터베이스당 하나의 영속성 유닛을 사용한다.

⑤ 테스트를 위한 클라이언트 프로그램 작성

■ src/main/java/BoardServiceClient.java

```
1   package tommy.spring.board;

2   import java.util.List;

3   import javax.persistence.EntityManager;
4   import javax.persistence.EntityManagerFactory;
5   import javax.persistence.EntityTransaction;
6   import javax.persistence.Persistence;

7   public class BoardServiceClient {
8           public static void main(String[] args) {
                    // EntityManager 생성
9                   EntityManagerFactory factory =
                            Persistence.createEntityManagerFactory("JPAProject");
10                  EntityManager manager = factory.createEntityManager();

                    // Transaction 생성
11                  EntityTransaction tx = manager.getTransaction();

12                  try {
                            // Transaction 시작
13                          tx.begin();

14                          Board board = new Board();
```

```
15                              board.setTitle("JPA 테스트");
16                              board.setWriter("손오공");
17                              board.setContent("JPA 글 등록 테스트");

                                // 글 등록
18                              manager.persist(board);

                                // 글 목록 조회
19                              String jpasql = "select m from Board m order by m.seq desc";
20                              List<Board> boardList = manager.createQuery(jpasql,
                                                                Board.class).getResultList();

21                              for (Board myBoard : boardList) {
22                                      System.out.println("---> : " + myBoard.toString());
23                              }

                                // Transaction Commit
24                              tx.commit();
25                      } catch (Exception e) {
26                              e.printStackTrace();
                                // Transaction RollBack
27                              tx.rollback();
28                      } finally {
29                              manager.clear();
30                      }
31                      factory.close();
32              }
33      }
```

⑥ 실행 결과 확인 및 데이터베이스 테이블 확인

	SEQ	CNT	CONTENT	REGDATE	TITLE	WRITER
1	1	0	JPA 글 등록 테스트	20/07/12	JPA 테스트	손오공

```
Hibernate:

    drop table MYBOARD cascade constraints
7월 12, 2020 6:43:45 오전 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@42c2f48c] for (non-JTA) DDL execut
Hibernate:

    drop sequence hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
7월 12, 2020 6:43:45 오전 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@2a49fe] for (non-JTA) DDL executic
Hibernate:

    create table MYBOARD (
       seq number(10,0) not null,
        cnt number(10,0) not null,
        content varchar2(255 char),
        regDate date,
        title varchar2(255 char),
        writer varchar2(255 char),
        primary key (seq)
    )
7월 12, 2020 6:43:45 오전 org.hibernate.engine.transaction.jta.platform.internal.JtaPlatformInitiator initiateService
INFO: HHH000490: Using JtaPlatform implementation: [org.hibernate.engine.transaction.jta.platform.internal.NoJtaPlatform]
Hibernate:
    select
        hibernate_sequence.nextval
    from
        dual
Hibernate:
    insert
    into
        MYBOARD
        (cnt, content, regDate, title, writer, seq)
    values
        (?, ?, ?, ?, ?, ?)
Hibernate:
    select
        board0_.seq as seq1_0_,
        board0_.cnt as cnt2_0_,
        board0_.content as content3_0_,
        board0_.regDate as regdate4_0_,
        board0_.title as title5_0_,
        board0_.writer as writer6_0_
    from
        MYBOARD board0_
    order by
        board0_.seq desc
---> : Board [seq=1, title=JPA 테스트, writer=손오공, content=JPA 글 등록 테스트, regDate=Sun Jul 12 06:43:45 KST 2020, cnt=0]
7월 12, 2020 6:43:46 오전 org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PoolState stop
INFO: HHH10001008: Cleaning up connection pool [jdbc:oracle:thin:@localhost:1521/XEPDB1]
```

■ 참고 : 여기에서 주의할 점은 쿼리문 작성인데 select m from Board m order by m.seq desc 테이블 명이 myboard가 아니고 Board라는 점이다. 현재 작업하는 테이블 명은 myboard이고 myboard 테이블과 매핑이된 클래스(@Table(name="MYBOARD")는 Board.java이고 클래스 명은 Board이다. 즉 쿼리문을 작성할때는 MYBOARD가 아닌 Board(클래스 명)를 입력해야 한다. 또한 대소문자를 구별하니 대소문자에도 유의해야 한다.

3. 영속성 유닛 설정

① 영속성 유닛 이름 지정

▢ 영속성 유닛은 연동할 데이터베이스당 하니 씩 등록하며 영속성 유닛에 설정된 이름은 나중에 DAO 클래스를 구현할 때 EntytyManagerFactory 객체 생성에 사용된다.

persistence Unit 설정	<persistence-unit name="JPAProject"></persistence-unit>
Java 소스	EntityManagerFactory factory = Persistence.createEntityManagerFactory("JPAProject"); EntityManager manager = factory.createEntityManager();

② 엔티티 클래스 등록

▢ 엔티티 클래스는 JPA 프로젝트에서 JPA Entity 클래스를 작성하는 순간 자동으로 persistence.xml 파일에 등록된다.

```
<persistence-unit name="JPAProject">
        <class>tommy.spring.board.Board</class>
</persistence-unit>
```

▢ 스프링 프레임워크나 J2EE환경에서 JPA를 사용한다면 자동으로 엔티티 클래스를 검색하여 처리하는 기능이 제공되므로 엔티티 클래스들을 일일이 등록할 필요는 없다.

▢ 하지만 JPA를 단독으로 사용하는 경우라면 엔티티 클래스를 등록하는 것이 가장 확실한 방법이다.

③ 영속성 유닛 프로퍼티 설정

```
<property name="javax.persistence.jdbc.driver"
value="oracle.jdbc.driver.OracleDriver"/>
<property name="javax.persistence.jdbc.user" value="mytest"/>
<property name="javax.persistence.jdbc.password" value="mytest"/>
<property name="javax.persistence.jdbc.url"
value="jdbc:oracle:thin:@localhost:1521/XEPDB1"/>
```

▢ 프로퍼티 설정에서 가장 중요하면서 기본적인 것이 DB 커넥션 관련 설정이다.

▢ 여기서 설정된 DB 커넥션 정보를 바탕으로 하이버네이트와 같은 JPA 구현체가 특정 데이터베이스와 커넥션을 연결할 수 있기 때문이다.

▢ 하지만 스프링 프레임워크와 연동할 때는 데이터소스가 스프링 설정파일에 등록되어 있으므로 영속성 유닛 설정에서는 제거될 수 도 있다.

프로퍼티 이름	프로퍼티 의미
javax.persistence.jdbc.driver	JDBC 드라이버 클래스
javax.persistence.jdbc.user	데이터베이스 접속 아이디
javax.persistence.jdbc.password	데이터베이스 접속 패스워드
javax.persistence.jdbc.url	JDBC URL 정보

④ Dialect 클래스 설정

```
<property name="hibernate.dialect"
value="org.hibernate.dialect.Oracle9Dialect"/>
```

▢ ORM 프레이워크의 가장 중요한 특징이자 장점은 애플리케이션을 수행할 때 필요한 SQL 구문을 자동으로 생성한다는 것이다. 그런데 문제는 이 SQL문을 아무리 표준에 따라서 잘 작성한다고 해도 DBMS에 따라서 키 생성 방식도 다르고 지원되는 함수도 다르다. 따라서 DBMS가 변경되면 이런 세세한 부분까지 개발자가 적절하게 수정해야 한다.

▢ JPA는 특정 DBMS에 최적화된 SQL문을 제공하기 위해서 DBMS 마다 다른 Dialect 클래스를 제공한다.

▢ Dialect는 사투리 또는 방언이라는 뜻인데 이 Dialect 클래스가 해당 DBMS에 최적화된 SQL 구문을 생성하는 것이다. DBMS가 변경되는 경우 Dialect 클래스만 변경하면 SQL 구문이 자동으로 변경되어 생성되므로 유지보수 성능이 크게 향상될 것이다.

▢ Dialect 클래스들의 정확한 위치는 Maven Dependecy에서 'hibernate-core-5.4.18.Final.jar' 파일이다. 현존하는 거의 대부분 관계형 데이터베이스에 해당하는 Dialect 클래스들이 제공된다.

데이터베이스	Dialect 클래스
DB2	org.hibernate.dialect.DB2Dialect

PostgreSQL	org.hibernate.dialect.PostgreDialect
MySQL	org.hibernate.dialect.MySQLDialect
Oracle	org.hibernate.dialect.Oracle9Dialect
Sybase	org.hibernate.dialect.SybaseDialect
Microsoft SQL Server	org.hibernate.dialect.SQLServerDialect
SAP DB	org.hibernate.dialect.SAPDBDialect
H2	org.hibernate.dialect.H2Dialect

⑤ JPA 구현체 관련 속성 설정

```
<property name="hibernate.show_sql" value="true"/>
<property name="hibernate.format_sql" value="true"/>
<property name="hibernate.use_sql_comments" value="false"/>
<property name="hibernate.id.new_generator_mappings" value="true"/>
<property name="hibernate.hbm2ddl.auto" value="create"/>
```

▢ 마지막으로 하이버네이트 관련된 속성 설정이 들어가는데 이것은 우리가 사용한 JPA 구현체가 하이버네이트 프레임워크이기 때문이다.

속성	의미
hibernate.show_sql	생성된 SQL을 콘솔에 출력한다.
hibernate.format_sql	SQL을 출력할 때 일정한 포맷으로 보기 좋게 출력한다.
hibernate.use_sql_comments	SQL에 포함된 주석도 같이 출력한다.
hibernate.id.new_generator_mappings	새로운 키 생성 전략을 사용한다.
hibernate..hbm2ddl.auto	테이블 생성이나 수정 삭제 같은 DDL 구문을 자동으로 처리할지 지정한다.

▢ DDL 명령어와 관련된 hibernate..hbm2ddl.auto 설정

속성값	의미
create	애플리케이션을 실행할 때 기존 테이블을 삭제하고 엔티티 클래스에 설정된 새로운 테이블을 생성한다.(DROP -> CREATE)
create-drop	create와 기본적으로 같지만 애플리케이션이 종료되기 직전에 생성된 테이블을 삭제한다.(DROP -> CREATE -> DROP)
update	기존에 사용중인 테이블이 있으면 새 테이블을 생성하지 않고 재사용한다. 만약 엔티티 클래스의 매핑 설정인 변경되면 변경된 내용을 반영한다.(ALTER)

▢ 앞에 예제에서 기존에 작업하던 내용이 삭제되고 새롭게 데이터가 하나 들어간 이유는 우리가 이 설정(hibernate..hbm2ddl.auto)을 'create'로 하였기 때문이다.

▢ 또 콘솔에서 보면 hibernate_sequence 시퀀스가 자동으로 삭제되고 다시 생성되는 메시지를 확인할 수 있다.

▢ DDL 자동 생성 기능은 애플리케이션 실행 시점에 테이블이 자동으로 생성되므로 굉장히 편리해 보인다. 하지만 일반적으로 프로젝트 초기에 데이터 모델링이 마무리 되고 나서 비즈니스 컴포넌트 개발에 들어가므로 이 기능을 사용할 일은 거의 없을 것이다.

4. 엔티티 클래스 기본 매핑

JPA의 기본은 엔티티 클래스를 기반으로 관계형 데이터베이스에 저장된 테이터를 관리하는 것이다.

엔티티 클래스를 작성하는데 규칙은 public 클래스로 만들어야하고 default constructor가 반드시 있어야 한다. 이 외에는 특별한 규칙이 없다. 따라서 우리는 그냥 Value Object처럼 POJO 형태의 클래스로 작성하면 된다.

엔티티 매핑에서 가장 중요하고 복잡한 설정은 연관매핑 설정이다. 이러한 설정은 전문적인 지식이 요구되므로 여기서는 가장 기본적인 요소만 살펴보자.

① @Entity, @Id

▢ @Entity는 특정 클래스를 JPA가 관리하는 엔티티 클래스로 인식하는 가장 중요한 어노테이션이다.

▢ 엔티티 클래스 선언부 위에 @Entity를 붙이면 JPA가 이 클래스를 엔티티 클래스로 인식하여 관련된 테이블과 자동으로 매핑 처리한다.

▢ 엔티티 클래스와 매핑되는 테이블은 각 ROW를 식별하기 위한 PK 컬럼을 가지고 있다. 이런 테이블과 매핑되는 엔티티 클래스 역시 PK 컬럼과 매핑될 변수를 가지고 있어야 한다. 이러한 변수를 식별자라고 하며 식별자 필드는 엔티티 클래스라면 무조건 가지고 있어야 하고 @Id 어노테이션을 이용하여 선언한다.

```
12 @Entity
13 @Table(name = "MYBOARD")
14 public class Board {
15
16     @Id
17     @GeneratedValue
18     private int seq;
19     private String title;
20     private String writer;
21     private String content;
22     @Temporal(TemporalType.DATE)
23     private Date regDate = new Date();
24     private int cnt;
```

▢ @Entity가 추가된 Board 클래스는 BOARD 테이블과 자동으로 매핑 된다. 만약 BOARD 클래스와 다른 테이블을 매핑하려면 @Table을 사용해야 한다.

② @Table

▢ 엔티티 클래스를 정의할 때 엔티티 클래스와 매핑되는 테이블의 이름을 별도로 지정하지 않으면 엔티티 클래스 이름과 같은 이름의 테이블이 자동으로 매핑 된다.

▢ 만약 엔티티 클래스 이름과 다른 이름의 테이블을 매핑해야 될 경우에 @Table 어노테이션을 사용한다.

```
@Entity
@Table(name = "MYBOARD")
public class Board {
```

속성	의미
name	매핑될 테이블 이름을 지정한다.
catalog	데이터베이스 카탈로그(catalog)를 지정한다.
schema	데이터베이스 스키마(schema)를 지정한다.

uniqueConstraints	결합 unique 제약조건을 지정하며 여러 개의 컬럼이 결합되어 유일성을 보장해야 하는 경우에 사용한다.
	@Table(name = "MYBOARD",
	uniqueConstraints={
	@UniqueContraints(columnNames={"SEQ", "WRITER"})
	}
	)

③ @Column

☐ @Column은 엔티티 클래스의 변수와 테이블의 컬럼을 매핑할 때 사용한다. 일반적으로 엔티티 클래스의 변수이름과 컬럼 이름이 다를 경우 사용한다. 생략할 경우 기본적으로 변수이름과 같은 컬럼 이름이 매핑된다.

속성	의미
name	컬럼 이름을 지정한다.(생략 시 프로퍼티명과 동일하게 매핑)
unique	unique 제약조건을 추가한다. (기본값 : false)
nullable	null 상태 허용여부를 지정한다. (기본값 : false)
insertable	입력 SQL 명령어를 자동으로 생성할 때 이 컬럼을 포함할 것인지를 지정한다. (기본값 : true)
updatalble	수정 SQL 명령어를 자동으로 생성할 때 이 컬럼을 포함할 것인지를 지정한다. (기본값 : true)
columnDefinition	이 컬럼에 대한 DDL문을 직접 설정한다.
length	문자열 타입의 컬럼 길이를 지정한다. (기본값 : 255)
precision	숫자 타입의 전체 자리수를 지정한다. (기본값 : 0)
scale	숫자 타입의 소수점 자리수를 지정한다. (기본값 : 0)

■ 적용 예시

```
@Column(name="BOARD_TITLE", nullable=false, length=30)
private String title;
@Column(name="BOARD_WRITER", updatable=false)
private String writer;
@Column(name="BOARD_CONTENT", nullable=false)
private String content;
```

④ @GeneratedValue

☐ @GeneratedValue는 @Id로 지정된 식별자 필드에 Primary Key 값을 생성하여 저장할 때 사용한다.

```
@Entity
@Table(name = "MYBOARD", uniqueConstraints={@UniqueContraints(columnNames={"SEQ", "WRITER"}
public class Board {

    @Id
    @GeneratedValue(strategy=GenerationType.SEQUENCE)
    private int seq;
    @Column(name="BOARD_TITLE", nullable=false, length=30)
    private String title;
```

☐ @GeneratedValue 속성

속성	의미
strategy	자동 생성 유형을 지정한다 (GenerationType 지정)
generator	이미 생성된 Generator 이름을 지정한다.

☐ 위의 속성 중에서 strategy는 PK 값 생성 전략을 지정하는 속성으로 중요하다.

■ PK 값 생성 전략

속성	전략
TABLE	Hibernate가 데이터 베이스 테이블을 사용하여 PK값을 생성한다. 따라서 PK값 생성을 위한 별도의 테이블이 필요하다.
SEQUENCE	Sequence Object를 이용하여 PK값을 생성한다. 이 전략은 Oracle과 같은 Sequence를 지원하는 데이터베이스에서만 사용할 수 있다.
IDENTITY	auto_increment나 IDENTITY를 이용하여 PK를 생성한다. 일반적으로 MySQL 같은 데이터베이스를 이용할 때 사용한다.
AUTO	Hibernate가 사용 중인 데이터베이스에 맞게 자동으로 PK 값을 생성한다. 아무런 설정이 없을 경우 이 값을 기본 값으로 사용한다.

⑤ @Transient

☐ 엔티티 클래스의 변수들은 대부분 테이블의 컬럼과 매핑 된다. 그러나 몇몇 변수는 매핑 되는 컬럼이 없거나 아예 매핑에서 제외해야 하는 경우가 발생한다.

☐ @Transient는 엔티티 클래스 내의 특정 변수를 영속 필드에서 제외할 때 사용한다.

```
@Entity
@Table(name = "MYBOARD")
public class Board {

    @Transient
    private String searchCondition;
    @Transient
    private String searchKeyword;
```

⑥ @Temporal

▢ @Temporal은 java.util.Date 타입의 날짜 데이터를 매핑할 때 사용한다.

■ TemporalType을 지정하면 출력되는 날짜의 형식을 지정할 수 있다.

	의미
TemporalType.DATE	날짜 정보만 출력한다.
TemporalType.TIME	시간 정보만 출력한다.
TemporalType.TIMESTAMP	날짜와 시간정보 모두 출력한다.

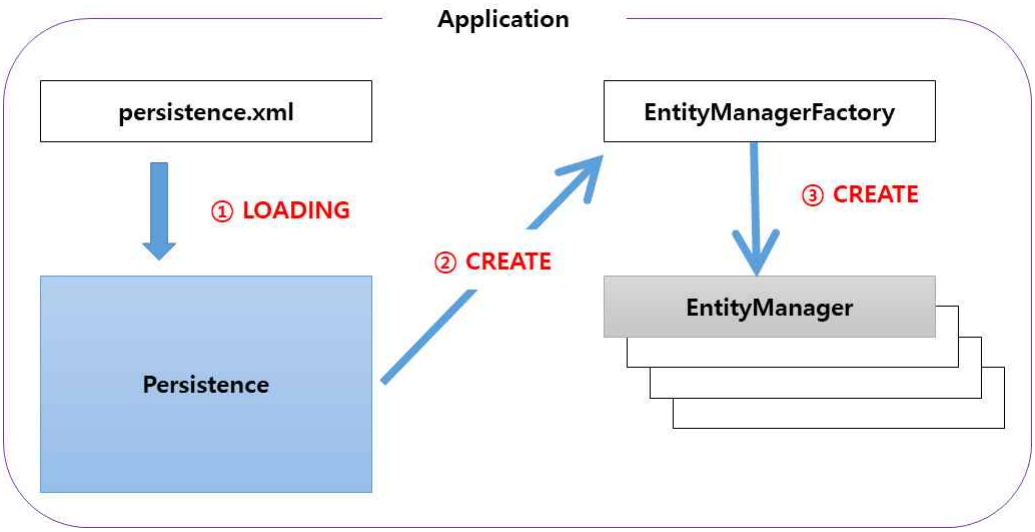
```
private String content;
@Temporal(TemporalType.DATE)
private Date regDate = new Date();
private int cnt;
```

5. JPA API

① JPA API 구조

▢ 엔티티 클래스에 기본적인 매핑을 설정했으면 이제 JPA에서 지원하는 API를 이용하여 데이터베이스 연동을 처리할 수 있다.

▢ 애플리케이션에서 JPA를 이용하여 CRUD 기능을 처리하려면 엔티티 관리자(EntityManager) 객체를 사용해야 한다. 그런데 이 EntityManager 객체는 EntityManagerFactory로부터 얻어낸다. 따라서 JPA를 이용한 애플리케이션의 사작은 EntityManager 객체의 생성이라고 할 수 있다.



① Persistence 클래스를 이용하여 영속성 유닛(persistence-unit) 정보가 저장된 JPA 메인 환경파일(persistence.xml)을 로딩한다.

② 이 설정 정보를 바탕으로 EntityManager를 생성하는 공장 기능의 EntityManagerFactory 객체를 생성한다.

③ 이제 EntityManagerFactory로부터 필요한 EntityManager를 얻어서 사용하면 된다.

▢ EntityManagerFactory 객체를 생성할때는 영속성 유닛이 필요하므로 persistence.xml 파일에 설정한 영속성유닛(persistence-unit) 이름을 지정하여 EntityManagerFactory 객체를 생성한다.

```
// EntityManager 생성
EntityManagerFactory factory = Persistence.createEntityManagerFactory("JPAProject");
EntityManager manager = factory.createEntityManager();
```

▢ EntityManagerFactory로부터 EntityManager 객체를 얻었으면 EntityManager를 통해서 EntityTransaction 객체를 얻을 수 있다. 이렇게 얻은 EntityTransaction을 통해 트랜잭션을 제어할 수 있다.

```
// Transaction 생성
EntityTransaction tx = manager.getTransaction();
```

▢ EntityManager가 제공하는 CRUD 기능의 메서드

메서드	기능설명
persist(Object entity)	엔티티를 영속한다.(INSERT)
merge(Object entity)	준영속 상태의 엔티티를 영속한다(UPDATE)
remove(Object entity)	영속상태의 엔티티를 제거한다(DELETE)
find(Class<T> entityClass, Object primaryKey)	하나의 엔티티를 검색한다(SELECT ONE)
createQuery(String query, Class<T> resultClass)	JPQL에 해당하는 엔티티 목록을 검색한다. (SELECT LIST)

② JPA API 사용

▢ 앞에서 작성한 BoardServiceClient 소스코드를 분석해 보자.

```
// Transaction 시작
tx.begin();

Board board = new Board();
board.setTitle("JPA 테스트");
board.setWriter("손오공");
board.setContent("JPA 글 등록 테스트");

// 글 등록
manager.persist(board);
```

▢ 위 코드를 보면 먼저 트랜잭션을 시작하고 엔티티 클래스로 등록된 Board 객체를 생성한 다음 글 등록에 필요한 값들을 저장한다. 여기에서 중요한 것은 단순히 엔티티 객체를 생성하고 여기에 값을 저장했다고 해서 이 객체가 MYBOARD 테이블과 자동으로 매핑되지 않는다는 것이다.

▢ 반드시 EntityManager의 persist()메서드로 엔티티 객체를 영속화 해야지만 INSERT 작업이 처리된다.

▢ 게시글이 등록되었으면 글 목록을 조회하는데 이때는 JPQL(Java Persistence Query Language)이라는 JPA 고유의 쿼리 구문을 사용해야 한다.

```
// 글 목록 조회
String jpasql = "select m from Board m order by m.seq desc";
List<Board> boardList = manager.createQuery(jpasql, Board.class).getResultList();

for (Board myBoard : boardList) {
    System.out.println("---> : " + myBoard.toString());
}

// Transaction Commit
tx.commit();
```

☐ JPQL은 기존에 사용하던 SQL과 거의 유사한 문법을 제공하므로 해석하는데는 별 어려움이 없다. 하지만 검색 대상이 테이블이 아닌 객체라는 점에 작성하는데 유의해야 한다.

☐ JPQL을 작성하고 실행하면 하이버네이트 같은 JPA 구현체가 JPQL을 연동되는 DBMS에 맞게 적절한 SELECT 구문으로 변환한다.

☐ 데이터베이스 연동 처리중 예외가 발생한다면 catch 블록에서 트랜잭션을 rollback 처리하면 되고 finally 블록에서 반드시 EntityManager를 close() 메서드를 이용해서 닫아야 한다.

☐ 그리고 프로그램이 종료되기 전에 EntitymanagerFactory 객체도 close() 메서드로 닫아야 한다.

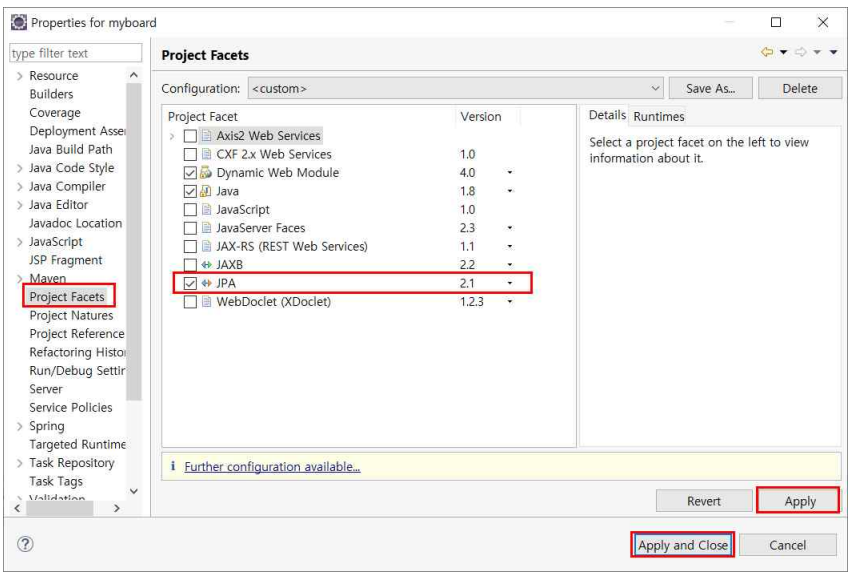
```
        } catch (Exception e) {
                e.printStackTrace();
                // Transaction RollBack
                tx.rollback();
        } finally {
                manager.clear();
            }
            factory.close();
        }
    }
```

6 스프링과 JPA 연동

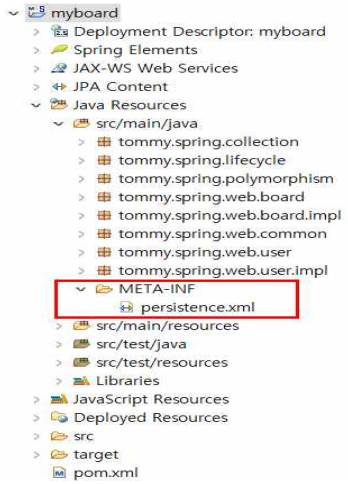
① 프로젝트 변경

☐ 스프링과 JPA를 연동하려면 우선 myboard 프로젝트를 JPA 프로젝트로 변환해야 한다. 기존에 사용하던 myboard 프로젝트를 선택하고 마우스 우 클릭하여 [Prperties]탭을 클릭한다.

☐ 왼쪽의 [Project Facets]른 선택하고 'Configuration' 항목에서 'JPA'를 체크한다.



▢ [Apply and Close]를 하고 나면 myboard 프로젝트가 JPA 프로젝트로 변환되고 아래와 같이 META-INF 폴더 안에 persistence.xml 파일이 자동으로 생성될 것이다.



② 라이브러리 내려받기

▢ 프로젝트가 변경되었으면 mvnrepository에 가서 hibernate-entitymanager, spring-orm을 검색하여 라이브러리를 추가하도록 한다.

```
<!-- hibernate-entitymanager -->
<dependency>
    <groupId>org.hibernate</groupId>
    <artifactId>hibernate-entitymanager</artifactId>
    <version>5.4.18.Final</version>
</dependency>
<!-- spring-orm -->
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-orm</artifactId>
    <version>5.1.9.RELEASE</version>
</dependency>
```

③ JPA 설정 파일 작성

▢ 라이브러리를 추가하였으면 아래와 같이 persistence.xml 파일을 작성하도록 하자.

```
<!--상단부분 생략 -->
    <persistence-unit name="myboard">
1       <class>tommy.spring.web.board.BoardVO</class>
2       <properties>
3           <!-- 필수 속성 -->
            <property name="hibernate.dialect"
4               value="org.hibernate.dialect.Oracle9Dialect" />
            <!-- 옵션 -->
5           <property name="hibernate.show_sql" value="true" />
6           <property name="hibernate.format_sql" value="true" />
```

```
7                                   <property name="hibernate.use_sql_comments" value="false" />
8                                   <property name="hibernate.id.new_generator_mappings"
                                            value="true" />
9                                   <property name="hibernate.hbm2ddl.auto" value="create" />
10                          </properties>
11                  </persistence-unit>
12          </persistence>
```

▢ 위 설정을 보면 데이터베이스 커넥션 관련 설정이 모두 삭제되어 있는데 이는 JPA를 스프링과 연동하면 스프링에서 제공하는 데이터소스를 이용해서 작업을 수행하면 되기 때문이다.

④ 엔티티 배핑 설정

▢ BoardVO 클래스를 JPA가 제공한느 어노테이션으로 엔티티 매핑을 설정한다.

▢ 이때 Spring MVC 학습에서 XML 변환 처리에 사용했던 JAXB2 관련 어노테이션은 모두 제거한다.

■ BoardVO 클래스 수정

```
1   package tommy.spring.web.board;

2   import java.util.Date;

3   import javax.persistence.Entity;
4   import javax.persistence.GeneratedValue;
5   import javax.persistence.Id;
6   import javax.persistence.Table;
7   import javax.persistence.Temporal;
8   import javax.persistence.TemporalType;
9   import javax.persistence.Transient;

10  import org.springframework.web.multipart.MultipartFile;

11  @Entity
12  @Table(name = "MYBOARD")
13  public class BoardVO {
14          @Id
15          @GeneratedValue
16          private int seq;
17          private String title;
18          private String writer;
19          private String content;
20          @Temporal(TemporalType.DATE)
21          private Date regDate;
22          private int cnt;
23          @Transient
24          private String searchCondition;
25          @Transient
26          private String searchKeyword;
27          @Transient
28          private MultipartFile uploadFile;
```

```
        // getter, setter 추가
        // Generate toString() 추가
```

▢ 엔티티 클래스 이름과 테이블 이름이 달라서 클래스 위에 @Table 어노테이션을 추가 했다.
▢ SEQ 컬럼과 매핑되는 seq 변수에 @Id와 @GeneratedValue를 사용하여 seq 변수를 식별자 필드로 지정함과 동시에 시퀀스를 이용하여 자동으로 값이 증가하도록 하였다.
▢ regDate 변수에는 시간을 제외한 날짜 정보만 저장되도록 @Temporal을 설정 하였다.
▢ searchCondition, searchKeyword, uploadFile 세 개의 변수에는 @Transient 어노테이션을 설정하여 영속필드에 제외하였다.

⑤ 스프링과 JPA 연동 설정

▢ 스프링과 JPA 연동을 위해 첫 번째로 JpaVendorAdapter 클래스를 등록한다. JpaVendorAdapter 클래스는 실제로 DB연동에 사용할 JPA 벤터를 지정할 때 사용하는데 우리는 하이버네이트를 JPA 구현체로 사용하고 있으므로 JpaVendorAdapter 클래스로 HibernateJpaVendorAdapter를 <bean>으로 등록하면 된다.
▢ 두 번째로 EntityMangerFactoryBean 클래스를 등록한다. JPA를 이용하여 DAO 클래스를 구현하려면 최종적으로 EntityManager 객체가 필요하다.
이 EntityManager 객체를 생성하려면 LocalContainerEntityManagerFactoryBean 클래스를 <bean>으로 등록해야 한다. 이때 앞에서 설정한 DataSource와 JpaVendorAdapter를 의존성 주입해 주면 된다.

■ applicationContext.xml 파일 수정

```
<!--상단부분 생략 -->
<!-- 스프링과 JPA 연동 설정 -->
<bean id="jpaVendorAdapter"
                class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
</bean>

<bean id="entityManagerFactory"
                class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
        <property name="dataSource" ref="dataSource"></property>
        <property name="jpaVendorAdapter" ref="jpaVendorAdapter"></property>
</bean>
<!--하단부분 생략 -->
```

▢ 참고로 만약 위 설정에서 LocalContainerEntityManagerFactoryBean 클래스를 <bean>으로 등록할 때 아래와 같이 영속성 유닛 관련된 설정을 같이 처리한다면 persistence.xml 파일을 작성하지 않아도 된다.

```
<!--상단부분 생략 -->
<!-- 스프링과 JPA 연동 설정 -->
<bean id="jpaVendorAdapter"
                class="org.springframework.orm.jpa.vendor.HibernateJpaVendorAdapter">
</bean>
```

```
3  <bean id="entityManagerFactory"
                    class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
4          <property name="dataSource" ref="dataSource"></property>
5          <property name="jpaVendorAdapter" ref="jpaVendorAdapter"></property>
6          <property name="packagesToScan" value="tommy.spring.web.board"></property>
7          <property name="jpaProperties">
8                  <props>
9                          <prop key="hibernate.dialect">
10                                 org.hibernate.dialect.Oracle9Dialect
11                         </prop>
12                         <prop key="hibernate.show_sql">true</prop>
13                         <prop key="hibernate.format_sql">true</prop>
14                         <prop key="hibernate.use_sql_comments">true</prop>
15                         <prop key="hibernate.id.new_generator_mappings">true</prop>
16                         <prop key="hibernate.hbm2ddl.auto">create</prop>
17                 </props>
18         </property>
19 </bean>
   <!--하단부분 생략 -->
```

⑥ 트랜잭션 설정 수정

▢ 우리는 앞에서 트랜잭션 관리를 스프링 컨테이너에게 위임할 때 DataSourceTransactionManager 클래스를 <bean>으로 사용하였다. DataSourceTransactionManager 클래스는 Spring JDBC나 MyBatis를 이용하여 DB연동을 처리할 때 사용하는 트랜잭션 관리자 이다.

▢ JPA를 이용하여 DB연동을 처리하고자 하면 JpaTransactionManager로 변경해야 한다.

■ applicationContext.xml 파일 수정

```
   <!--상단부분 생략 -->
   <!-- 스프링과 JPA 연동 설정 추가 -->
1  <bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
2          <property name="entityManagerFactory" ref="entityManagerFactory"></property>
3  </bean>

   <!--
       <tx:advice>와 <aop:config>를 아래와 같이 수정하고, 이 부분을 presentation-layer.xml 로
       옮겨놓으면 된다.
   -->
4  <tx:advice id="txAdvice" transaction-manager="txManager">
5          <tx:attributes>
6                  <tx:method name="get*" read-only="true" />
7                  <tx:method name="*" />
8          </tx:attributes>
9  </tx:advice>

10 <aop:config>
11         <aop:pointcut
                   expression="execution(* tommy.spring.web..*Impl.*(..))" id="txPointcut" />
12         <aop:advisor advice-ref="txAdvice" pointcut-ref="txPointcut" />
```

```
13  </aop:config>
    <!--하단부분 생략 -->
```

⑦ DAO 클래스 구현

▢ JPA를 이용해서 DAO 클래스를 구현할 때는 EntityManager 객체를 사용해야 하는데 JPAProject에서는 EntityManagerFactory로부터 EntityManger 객체를 직접 얻어냈다.

▢ 하지만 JPA를 단독으로 사용하지 않고 스프링과 연동할때는 EntityManagerFactory에서 EntityManager를 직접 생성하는 것이 아니라 스프링 컨테이너가 제공하는 EntityManager를 사용해야만 한다.

■ BoardDAOJPA 클래스 작성

```
1   package tommy.spring.web.board.impl;

2   import java.util.List;

3   import javax.persistence.EntityManager;
4   import javax.persistence.PersistenceContext;

5   import org.springframework.stereotype.Repository;

6   import tommy.spring.web.board.BoardVO;

7   @Repository
8   public class BoardDAOJPA {

9           @PersistenceContext
10          private EntityManager manager;

11          public void insertBoard(BoardVO vo) {
12                  System.out.println("JPA로 insertBoard() 기능 처리");
13                  manager.persist(vo);
14          }

15          public void updateBoard(BoardVO vo) {
16                  System.out.println("JPA로 updateBoard() 기능 처리");
17                  manager.merge(vo);
18          }

19          public void deleteBoard(BoardVO vo) {
20                  System.out.println("JPA로 deleteBoard() 기능 처리");
21                  manager.remove(manager.find(BoardVO.class, vo.getSeq()));
22          }

23          public BoardVO getBoard(BoardVO vo) {
24                  System.out.println("JPA로 getBoard() 기능 처리");
25                  return (BoardVO) manager.find(BoardVO.class, vo.getSeq());
26          }
```

```
27          public List<BoardVO> getBoardList(BoardVO vo) {
28                  System.out.println("JPA로 getBoardList() 기능 처리");
29                  return manager.createQuery(
                            "select vo from BoardVO vo order by vo.seq desc", BoardVO.class)
                            .getResultList();
30          }
31  }
```

▢ @PersistenceContext는 스프링 컨테이너가 관리하는 EntityManager 객체를 의존성 주입할 때 사용하는 어노테이션이다.

⑧ BoardServiceImpl 클래스 수정 및 테스트

▢ BoardServiceimpl 클래스에서 BoardDAOMybatis 대신에 추가된 BoardDAOJPA 클래스로 변경하고 DB 연동을 처리하면 된다.

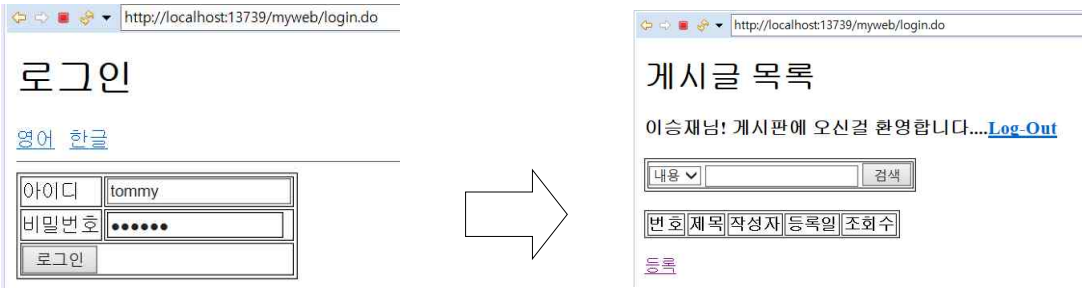
```
    <!--상단부분 생략 -->
1   @Service("boardService")
2   public class BoardServiceImpl implements BoardService {
3           @Autowired
4           private BoardDAOJPA boardDAO;
    <!--하단부분 생략 -->
```

■ 지금까지 수정한 모든 파일을 저장하고 게시판 프로그램을 실행해 보자.

■ 참고 : <prop key="hibernate.hbm2ddl.auto">create</prop>와 같이 설정하였으므로 시퀀스와 테이블이 삭제되고 다시 만들어 질 것이다.

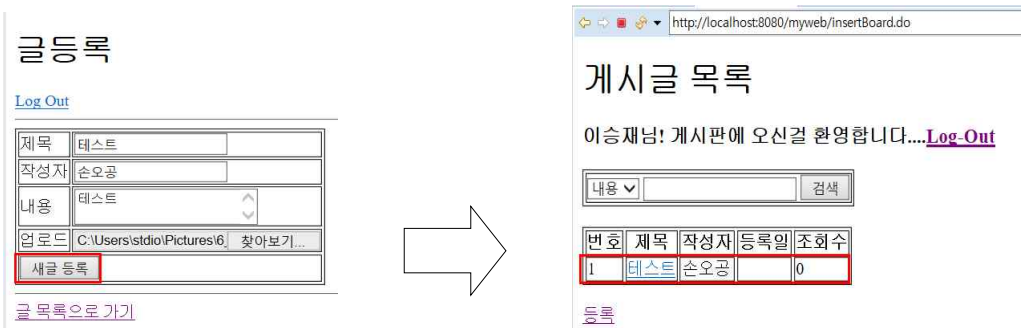
```
정보: Initializing Spring root WebApplicationContext
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext: initialization started
WARN : org.hibernate.dialect.Oracle9Dialect - HHH000063: The Oracle9Dialect dialect has been deprecated; use either Oracle9iDialect or Oracle10gDialect instead
Hibernate:

    drop table MYBOARD cascade constraints
Hibernate:

    drop sequence hibernate_sequence
Hibernate: create sequence hibernate_sequence start with 1 increment by 1
Hibernate:

    create table MYBOARD (
       seq number(10,0) not null,
        cnt number(10,0) not null,
        content varchar2(255 char),
        regDate date,
        title varchar2(255 char),
        writer varchar2(255 char),
        primary key (seq)
    )
INFO : org.springframework.web.context.ContextLoader - Root WebApplicationContext initialized in 8144 ms
```

■ 로그인 후 글 등록을 수행하면 하이버네이트가 생성한 다양한 SQL 쿼리 문을 볼 수 있을 것이다.




```
로그인 화면으로 이동
로그인 인증 처리
JDBC로 getUser() 기능 처리
글 목록 검색 처리
JPA로 getBoardList() 기능 처리
Hibernate:
    select
        boardvo0_.seq as seq1_0_,
        boardvo0_.cnt as cnt2_0_,
        boardvo0_.content as content3_0_,
        boardvo0_.regDate as regdate4_0_,
        boardvo0_.title as title5_0_,
        boardvo0_.writer as writer6_0_
    from
        MYBOARD boardvo0_
    order by
        boardvo0_.seq desc
```

■ 글 등록 작업을 수행하는 화면



```
글 등록 처리
JPA로 insertBoard() 기능 처리
Hibernate:
    select
        hibernate_sequence.nextval
    from
        dual
Hibernate:
    insert
    into
        MYBOARD
        (cnt, content, regDate, title, writer, seq)
    values
        (?, ?, ?, ?, ?, ?)
글 목록 검색 처리
JPA로 getBoardList() 기능 처리
Hibernate:
    select
        boardvo0_.seq as seq1_0_,
        boardvo0_.cnt as cnt2_0_,
        boardvo0_.content as content3_0_,
        boardvo0_.regDate as regdate4_0_,
        boardvo0_.title as title5_0_,
        boardvo0_.writer as writer6_0_
    from
        MYBOARD boardvo0_
    order by
        boardvo0_.seq desc
```

■ 나머지 수정과 삭제를 수행하여 결과를 확인해 보자.