



[한국ICT인재개발원] 스프링 프레임워크

## 5. 스프링 CRUD와 테스트

前) 광고데이터 분석 1년

前) IT강의 경력 2년 6개월

前) 머신러닝을 활용한 데이터 분석 프로젝트반 운영 1년

前) 리그오브 레전드 데이터 분석 등...

現) 국비반 강의 진행중

```
11 <java-version>1.8</java-version>
12 <org.springframework-version>5.0.7.RELEASE</org.
13 <org.aspectj-version>1.6.10</org.aspectj-versior
14 <org.slf4j-version>1.6.6</org.slf4j-version>
```

```
95 <groupId>javax.servlet</groupId>
96 <artifactId>javax.servlet-api</ar
97 <version>3.1.0</version>
98 <scope>provided</scope>
```

```
112 <!-- Test -->
113 <dependency>
114     <groupId>junit</groupId>
115     <artifactId>junit</artif
116     <version>4.12</version>
117     <scope>test</scope>
```

```
139 <version>3.5.1</version>
140 <configuration>
141     <source>1.8</source>
142     <target>1.8</target>
```

Pom.xml 내부 수치변경을 진행합니다.  
위 코드 라인과 실제 수정값을 보고 그대로 적용해주세요.



## 1. Spring TestContext Framework

[org.springframework](https://org.springframework) » [spring-test](https://spring-test)

Spring TestContext Framework

Last Release on Feb 16, 2021

spring-test 5.0.7.RELEASE버전



## 1. HikariCP

[com.zaxxer](https://com.zaxxer) » [HikariCP](https://HikariCP)

Ultimate JDBC Connection Pool

Last Release on Mar 3, 2021

hikariCP 4.0.3버전



## 1. Log4Jdbc Log4j2 JDBC 4

[org.bgee.log4jdbc-log4j2](https://org.bgee.log4jdbc-log4j2) » [log4jdbc-log4j2-jdbc4](https://log4jdbc-log4j2-jdbc4)

Log4Jdbc Log4j2 JDBC 4

Last Release on Dec 12, 2013

log4jdbc-log4j2-jdbc4 1.16버전  
(한 칸 아래 JDBC 4.1 과 혼동X)



## 1. Spring JDBC

[org.springframework](https://org.springframework) » [spring-jdbc](https://spring-jdbc)

Spring JDBC

Last Release on Feb 16, 2021

spring-jdbc 5.0.7.RELEASE버전



## 1. MyBatis

[org.mybatis](https://org.mybatis) » [mybatis](https://mybatis)

The MyBatis SQL mapper framework  
XML descriptor or annotations. E

Last Release on Oct 6, 2020

mybatis 3.5.6버전



## 1. Project Lombok

[org.projectlombok](https://org.projectlombok) » [lombok](https://lombok)

Spice up your java: Automatic

Last Release on Jan 28, 2021

Lombok 1.18.18버전



## 1. Spring Transaction

[org.springframework](https://org.springframework) » [spring-tx](https://spring-tx)

Spring Transaction

Last Release on Feb 16, 2021

spring-tx 5.0.7.RELEASE버전



## 1. MyBatis Spring

[org.mybatis](https://org.mybatis) » [mybatis-spring](https://mybatis-spring)

An easy-to-use Spring bridge fo

Last Release on Nov 14, 2020

mybatis-spring 2.0.6버전



## 1. MySQL Connector/J

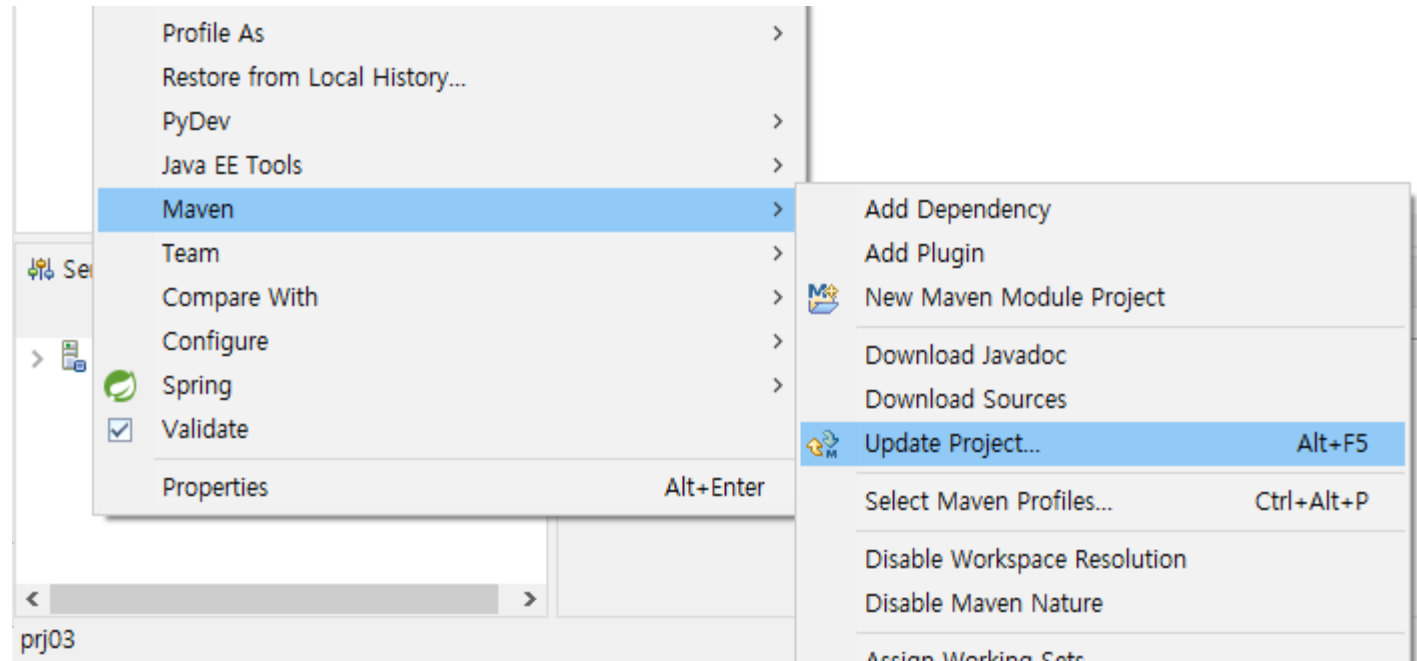
[mysql](https://mysql) » [mysql-connector-java](https://mysql-connector-java)

JDBC Type 4 driver for MySQL

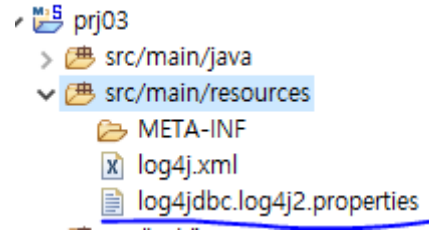
Last Release on Jan 17, 2021

mysql-connector-java 8.0.21  
(설치된 mysql버전과 동일하게)

Mvnrepository에서 검색어를 정확히 검색해 맞는 버전을  
Pom.xml의 dependencies에 추가해주세요



프로젝트 파일 우클릭 -> Maven -> Update Project를 클릭해 자바버전을 1.8버전으로 업데이트합니다.



```
log4jdbc.log4j2.properties  ⌕  
1 log4jdbc.spylogdelegator.name=net.sf.log4jdbc.log.slf4j.Slf4jSpyLogDelegator
```

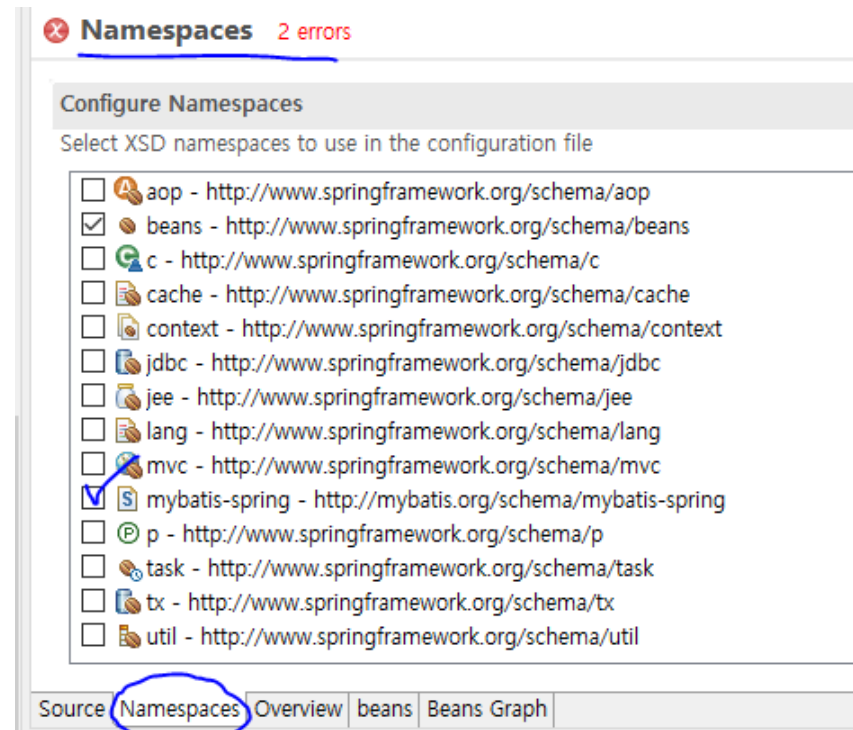
프로젝트의 src/main/resources 폴더에  
log4jdbc.log4j2.properties 라는 이름으로 폴더를 생성하고  
내용을 위와 같이 적습니다.

```
<bean id="hikariConfig" class="com.zaxxer.hikari.HikariConfig">
  <property name="driverClassName"
    value="net.sf.log4jdbc.sql.jdbcapi.DriverSpy"></property>
  <property name="jdbcUrl"
    value="jdbc:log4jdbc:mysql://localhost:3306/ict3?serverTimezone=UTC">
    </property>
  <property name="username" value="root"></property>
  <property name="password" value="1111"></property>
</bean>

<bean id="dataSource"
  class="com.zaxxer.hikari.HikariDataSource" destroy-method="close">
  <constructor-arg ref="hikariConfig"></constructor-arg>
</bean>
```

Root-context.xml로 이동해 hikariCP 관련 설정과 마이바티스 관련 설정을 진행합니다.

먼저 hikariCP의 HikariConfig와 HikariDataSource 객체를 bean-container에 넣어보겠습니다.

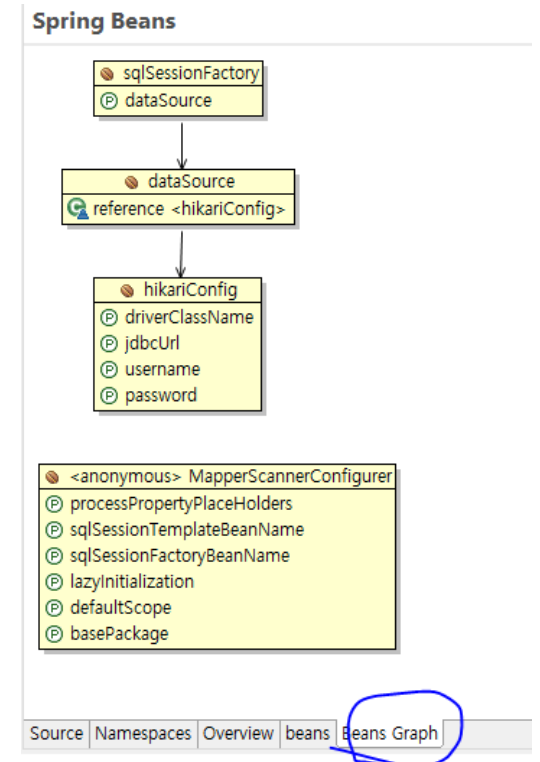


다음으로 root-context.xml 파일 하단의 Namespaces 탭을 클릭하고 mybatis-spring을 체크합니다.



```
<bean id="sqlSessionFactory" class="org.mybatis.spring.SqlSessionFactoryBean">
    <property name="dataSource" ref="dataSource"></property>
</bean>

<mybatis-spring:scan base-package="org.ict.mapper"/>
```



이후 위와 같이 `sqlSessionFactory` 객체를 생성한 다음 밑에 마이바티스 스캔을 하고, `root-context.xml` 하단의 `Beans Graph` 탭을 클릭해서 제대로 객체들이 들어왔나 체크합니다.

```
<!-- 한글 인코딩 설정 -->
<filter>
  <filter-name>encoding</filter-name>
  <filter-class>
    org.springframework.web.filter.CharacterEncodingFilter
  </filter-class>
  <init-param>
    <param-name>encoding</param-name>
    <param-value>UTF-8</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>encoding</filter-name>
  <servlet-name>appServlet</servlet-name>
</filter-mapping>
```

그리고, 한글인코딩을 처리하기 위해  
Web.xml 파일의 <web-app> 태그 사이에 위와 같이 적어넣습니다.  
일부 화면에 표시되는 한글, 그리고 폼에서 post로 전송되는 한글 처리를  
위해 이렇게 세팅해주면 됩니다.

```
create table ictboard(  
    bno int not null auto_increment,  
    content varchar(1000) not null,  
    title varchar(100) not null,  
    writer varchar(20) not null,  
    regdate timestamp default now(),  
    updatedate timestamp default now(),  
    primary key(bno)  
);
```

MySQL 테이블은 위와 같이 구성합니다.

글번호, 본문, 제목, 글쓴이, 등록날짜, 수정날짜를 가지며  
글번호를 프라이머리키로 가집니다.

Bno의 auto\_increment는 글 번호를 1부터 자동으로 insert구문 실행시마다  
1씩 증가시켜서 입력하라는 옵션입니다.

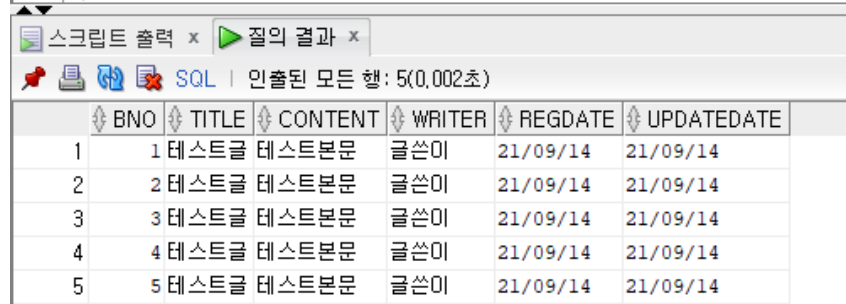
```
CREATE SEQUENCE board_num;  
  
CREATE TABLE board_tbl (  
    bno number(10, 0),  
    title varchar2(200) not null,  
    content varchar2(2000) not null,  
    writer varchar2(50) not null,  
    regdate date default sysdate,  
    updatedate date default sysdate  
);  
  
alter table board_tbl add constraint pk_board primary key(bno);
```

반면 오라클은 위와 같은 구문을 이용해 게시판 글 구성을 해 줍니다.

SEQUENCE는 일종의 변수로, auto\_increment가 없는 오라클의 특성상

```
insert into board_tbl (bno, title, content, writer) values (board_num.nextval, '테스트글', '테스트본문', '글쓴이');
```

```
select * from board_tbl;
```



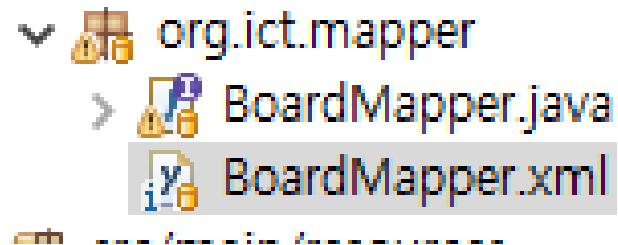
BNO	TITLE	CONTENT	WRITER	REGDATE	UPDATEDATE
1	1 테스트글	테스트본문	글쓴이	21/09/14	21/09/14
2	2 테스트글	테스트본문	글쓴이	21/09/14	21/09/14
3	3 테스트글	테스트본문	글쓴이	21/09/14	21/09/14
4	4 테스트글	테스트본문	글쓴이	21/09/14	21/09/14
5	5 테스트글	테스트본문	글쓴이	21/09/14	21/09/14

Insert 구문으로 5개의 글을 집어넣고 select 구문을 이용해 제대로 자료가 들어갔는지 체크합니다.

Bno는 보시다시피 시퀀스 설정으로 만든 board\_num.nextval 으로 입력하면 board\_num이 0인 상태로 시작해 nextval을 받을때마다 1씩 가산되어 입력됩니다.

기본 설정이 끝났습니다.

CRUD로직을 작성해보겠습니다.



```
BoardMapper.xml
1 <?xml version="1.0" encoding="UTF-8" ?>
2 <!DOCTYPE mapper
3     PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN"
4     "http://mybatis.org/dtd/mybatis-3-mapper.dtd">
5 <!-- 위의 스키마 작성 후, 어떤 인터페이스의 메서드를 쿼리문과
6 연결해 줄지 아래와 같이 인터페이스 파일부터 mapper태그로 연결합니다. -->
7 <mapper namespace="org.ict.mapper.BoardMapper">
8
```

```
<mybatis-spring:scan base-package="org.ict.mapper"/>
```

먼저, 마이바티스는 자바 코드와 쿼리문을 분리하기 위해 위와 같이 인터페이스와 xml파일 두 개의 나뉜 파일을 하나처럼 사용합니다.

이를 위해 root-context.xml 내부에 mybatis-spring:scan에 마이바티스 관련 파일이 적재되는 패키지를 인식시킵니다.

BoardMapper.java

```
public interface BoardMapper {  
  
    // board_tbl에서 글번호 3번 이하만 조회하는 쿼리문을  
    // 어노테이션을 이용해 작성해주세요.  
    //@Select("SELECT * FROM board_tbl WHERE bno < 4")  
    public List<BoardVO> getList();  
}
```

BoardMapper.xml

```
<select id="getList" resultType="org.ict.domain.BoardVO">  
    <!-- <이다 >과 같이 태그로 오인될 수 있는 문자를 포함하는  
    쿼리문은 그냥 작성하지 않고, CDATA를 이용합니다.  
    아래와 같이 <![CDATA[ 실행문 ]]> 과 같이 실행문을  
    CDATA내부에 작성하면 내부의 부등호는 전부 문자로 간주됩니다. -->  
    <![CDATA[  
        SELECT * FROM board_tbl WHERE bno < 4  
    ]]>  
</select>
```

위와 같이 인터페이스에서는 메서드의 이름과 용도에 따른 리턴자료형을 적어줄 수 있고,

Xml파일에서는 쿼리문의 종류에 따라 태그를 사용하고(위에서는 select 구문용 select 태그 사용중) id속성에 연결 메서드 이름을, resultType에는 리턴 자료형을 적되 리스트 자료형도 그냥 리스트 없이 자료만 적습니다.



BoardMapper.java

```
@Select("SELECT * FROM board_tbl WHERE bno < 4")  
public List<BoardVO> getList();
```

BoardMapper.xml

```
<select id="getList" resultType="org.ict.domain.BoardVO">  
  <!-- <에나 >과 같이 태그로 오인될 수 있는 문자를 포함하는  
  쿼리문은 그냥 작성하지 않고, CDATA를 이용합니다  
  아래와 같이 <![CDATA[ 실행문 ]]> 과 같이 실행문을  
  CDATA내부에 작성하면 내부의 부등호는 전부 문자로 간주됩니다. -->  
  <![CDATA[  
    SELECT * FROM board_tbl WHERE bno < 4  
  ]]>  
</select>
```

만약 위와 같이 인터페이스 쪽에서 어노테이션을 이용해 연결 쿼리문을 작성해주는 경우, xml쪽에서는 작성하지 말아야 합니다.

만약 양쪽 모두에 쿼리문을 연결하면 에러가 발생합니다.

BoardMapper.java

```
// insert구문 실행용으로 메서드를 선언합니다.  
// VO내부에 적혀있는 정보를 이용해 insert를 합니다.  
// BoardVO를 매개로 insert 정보를 전달받음.  
public void insert(BoardVO vo);
```

BoardVO.java

```
@Data  
public class BoardVO {  
  
    private Long bno;  
    private String title;  
    // 나머지도 작성해주세요  
    // 시간은 Date로 작성합니다. java  
    private String content;  
    private String writer;  
    private Date regdate;  
    private Date updatedate;  
}
```

BoardMapper.xml

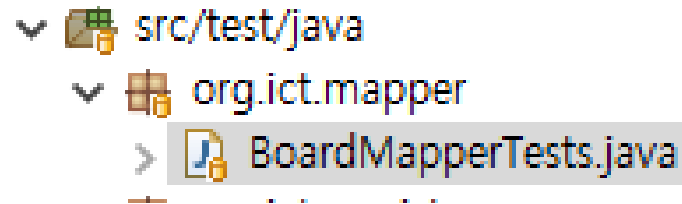
BoardMapper.java

```
// 글 번호(Long bno)를 파라미터로 받아  
// 해당 글 번호에 해당하는 글을 리턴해 보여주는 메서드를 작성해주세요.  
// 메서드 이름은 select 입니다.  
// xml파일에 쿼리문도 작성해보겠습니다.  
public BoardVO select(Long bno);
```

BoardMapper.xml

```
<!-- 방금 만든 select 메서드를 연결해보겠습니다.  
resultType은 리스트랑 개별자료형을 구분하지 않기 때문에  
그냥 BoardVO로 처리하면 됩니다. -->  
<select id="select" resultType="org.ict.domain.BoardVO">  
    SELECT * FROM board_tbl WHERE bno = #{bno}  
</select>
```

단일 변수를 생성하는 경우는 위와 같이 이름만 일치시키면 전달됩니다.



```
// 테스트코드 기본세팅(RunWith, ContextConfiguration, Log4j)해주세요.  
@RunWith(SpringJUnit4ClassRunner.class)  
@ContextConfiguration(  
    "file:src/main/webapp/WEB-INF/spring/root-context.xml")  
@Log4j  
public class BoardMapperTests {  
  
    // 이 테스트코드 내에서는 Mapper테스트를 담당합니다.  
    // 따라서 BoardMapper내부의 메서드를 실행할 예정이고  
    // BoardMapper 타입의 변수가 필요하니  
    // 선언해주시고 자동 주입으로 넣어주세요.  
    @Autowired  
    private BoardMapper mapper;
```

이제 테스트코드를 생성해 테스트 하는 부분을 보겠습니다.  
왼쪽과 같이 src/test/java 하위에 테스트 파일을 만들어주시면 되고  
의존성 주입을 통한 자동생성을 BoardMapper에 적용합니다.

BoardMapperTests.java

```
// 테스트용 메서드의 이름은 testGetList입니다.
// 테스트 코드가 실행될 수 있도록 만들어주세요.
// @Test
public void testGetList() {
    // mapper 내부의 getList 메서드를 호출하려면?
    log.info(mapper.getList());
}
```

BoardMapper.xml

```
<select id="getList" resultType="org.ict.domain.BoardVO">
    <!-- <이름>과 같이 태그로 오인될 수 있는 문자를 포함하는
    쿼리문은 그냥 작성하지 않고, CDATA를 이용합니다.
    아래와 같이 <![CDATA[ 실행문 ]]> 과 같이 실행문을
    CDATA내부에 작성하면 내부의 부등호는 전부 문자로 간주됩니다. -->
    <![CDATA[
        SELECT * FROM board_tbl WHERE bno < 4
    ]]>
</select>
```

일반 조회구문의 경우 위와 같이 미리 주입해둔 mapper내부의 메서드를 호출하면, xml이나 어노테이션으로 연결되어 있던 쿼리문이 호출되는 식으로 테스트가 진행됩니다.

BoardMapperTests.java

```
// insert를 실행할 테스트코드를 하단에 작성해보겠습니다.
//@Test
public void testInsert() {
    // 글 입력을 위해서 BoardVO 타입을 매개로 사용함
    // 따라서 BoardVO를 만들어놓고
    // setter로 글제목, 글본문, 글쓴이 만 저장해둔 채로
    // mapper.insert(vo);를 호출해서 실행여부를 확인하면 됨.
    // 위 설명을 토대로 아래 vo에 6번글에 대한 제목 본문 글쓴이를 넣고
    // 호출해주신 다음 실제로 DB에 글이 들어갔는지 확인해주세요.

    BoardVO vo = new BoardVO();

    // 입력할 글에 대한 제목, 글쓴이, 본문을 vo에 넣어줍니다.
    vo.setTitle("새로넣는글");
    vo.setContent("새로넣는본문");
    vo.setWriter("새로넣는글쓴이");

    //log.info(vo);
    mapper.insert(vo);
}
```

BoardMapper.xml

```
<!-- 현재 insert는 BoardVO를 파라미터로 받고 있습니다.
이 경우, VO 내부 변수를 #{변수명} 으로 쿼리문에 전달할 수 있습니다.
${변수명} 이 아님에 주의해주세요. -->
<insert id="insert">
    INSERT INTO board_tbl (bno, title, content, writer)
    VALUES
    (board_num.nextval, #{title}, #{content}, #{writer})
</insert>
```

글 삽입처럼 vo를 매개로 테스트해야하는 경우는 삽입구문 실행 이전에 먼저 vo를 생성해서 글에 필요한 정보를 입력해준 다음 Insert구문을 실행해 필요 데이터를 모두 전달해줍니다.

위 연결로직들을 응용해  
CRUD를 완성시키고 .jsp파일로  
뷰까지 완성시켜보겠습니다.