



# 09\_DB 연결하기

## 1. HiKariCP(Hikari Connection Pool) 란?

### 2. Spring Boot 프로젝트에 DB 연결하기

#### 2-1. application.properties 설정 방법

- 1) 오라클 DB와 연결 하는 객체 설정 (application.properties)
- 2) HikariCP를 이용해서 커넥션풀 생성 및 설정(application.properties)
- 3) 애플리케이션을 실행시켜 콘솔 확인

#### 2-2. @Configuration 클래스 파일을 이용한 방법

- 1) 오라클 DB와 연결 하는 객체 설정 (config.properties)
- 2) edu.kh.project.common.config.DBConfig 클래스 작성
- 3) 애플리케이션을 실행시켜 콘솔 확인

### 3. 마이바티스 추가 하기

#### 3-1. application.properties 설정 방법

application.properties 작성

#### 3-2. @Configuration 클래스 파일을 이용한 방법

edu.kh.project.common.config.DBConfig 클래스

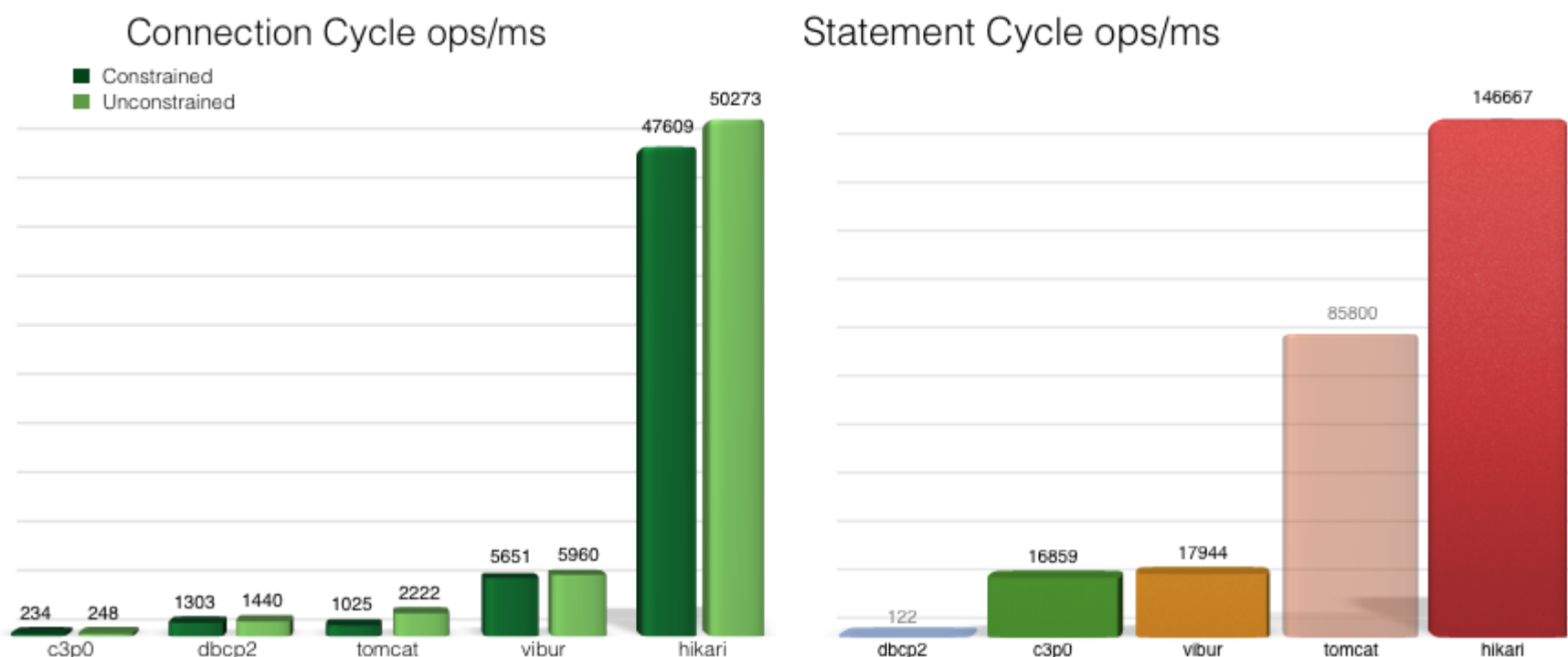
### 4. @Mapper 어노테이션

edu.kh.project.member.model.dao.MemberMapper.jav

src/main/resources/mappers/member-mapper.xml

## 1. HiKariCP(Hikari Connection Pool) 란?

- 빠르고, 간단하고 믿을 수 있는 HikariCP는 "오버헤드 제로(처리 시간을 극단적으로 줄이는 것을 목표)"의 프로덕션 지원 JDBC Connection Pool로 다른 Connection에 비해 매우 가벼움.
- 



HikariCP 성능 비교(benchmark) <https://github.com/brettwooldridge/HikariCP#artifacts>

build.gradle → dependencies에 추가

(Spring Starter 생성 시 JDBC API 추가하면됨)

```
// Spring Boot JDBC 관련 라이브러리 모음
implementation 'org.springframework.boot:spring-boot-starter-jdbc'

// 오라클 JDBC Driver
```

```
runtimeOnly 'com.oracle.database.jdbc:ojdbc8'

// @ConfigurationProperties 사용 가능
annotationProcessor 'org.springframework.boot:spring-boot-configuration-processor'
```

## 2. Spring Boot 프로젝트에 DB 연결하기

### 2-1. application.properties 설정 방법

#### 1) 오라클 DB와 연결 하는 객체 설정 (application.properties)

```
#오라클 DB 연결 정보
spring.datasource.hikari.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.hikari.jdbc-url=jdbc:oracle:thin:@IP주소:포트번호:DB이름
spring.datasource.hikari.username=계정명
spring.datasource.hikari.password=비밀번호

# HikariCP Connection Pool Properties
#풀이 연결을 사용할 수 있을 때까지 대기하는 최대 시간(ms)을 지정
spring.datasource.hikari.connection-timeout=30000

#풀의 최대 연결 수 설정
spring.datasource.hikari.maximum-pool-size=20

#연결이 풀에서 유휴 상태로 있을 수 있는 최대 시간(ms)을 지정
spring.datasource.hikari.idle-timeout=600000

#연결 풀의 이름을 지정
spring.datasource.hikari.pool-name=MyHikariCP

#자동 커밋 끄기
spring.datasource.hikari.auto-commit=false
```

#### [아래와 같은 오류가 나타나는 경우]

Failed to configure a DataSource: 'url' attribute is not specified and no embedded datasource could be configured.

```
spring.datasource.hikari.jdbc-url
→
spring.datasource.url 수정
```

#### 2) HikariCP를 이용해서 커넥션풀 생성 및 설정(application.properties)

```
# HikariCP Connection Pool Properties
#풀이 연결을 사용할 수 있을 때까지 대기하는 최대 시간(밀리초)을 지정합니다.
```

```
spring.datasource.hikari.connection-timeout=30000

#풀의 최대 연결 수 설정
spring.datasource.hikari.maximum-pool-size=50

# 연결이 풀에서 유휴 상태로 있을 수 있는 최대 시간(밀리초)을 지정합니다.
spring.datasource.hikari.idle-timeout=600000

#연결 풀의 이름을 설정합니다.
spring.datasource.hikari.pool-name=MyHikariCP
```

### 3) 애플리케이션을 실행시켜 콘솔 확인



```
:: Spring Boot :: (v3.1.0)

2023-05-23T16:10:20.668+09:00 INFO 11904 --- [ restartedMain] e.k.a.AdminprojectApplication : Starting AdminprojectApplication using Java 17.0.7 with PID 11904 (D:\work
2023-05-23T16:10:20.670+09:00 INFO 11904 --- [ restartedMain] e.k.a.AdminprojectApplication : No active profile set, falling back to 1 default profile: "default"
2023-05-23T16:10:20.712+09:00 INFO 11904 --- [ restartedMain] o.s.b.devtools.restart.ChangeableUrls : The Class-Path manifest attribute in C:\Users\knbdh\gradle\caches\modules
2023-05-23T16:10:20.712+09:00 INFO 11904 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to
2023-05-23T16:10:20.712+09:00 INFO 11904 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web
2023-05-23T16:10:21.184+09:00 INFO 11904 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-05-23T16:10:21.195+09:00 INFO 11904 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 4 ms. Found 0 JPA repository in
2023-05-23T16:10:21.283+09:00 WARN 11904 --- [ restartedMain] o.m.s.mapper.ClassPathMapperScanner : No MyBatis mapper was found in '[edu.kh.adminproject]' package. Please che
2023-05-23T16:10:21.643+09:00 INFO 11904 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 80 (http)
2023-05-23T16:10:21.652+09:00 INFO 11904 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-05-23T16:10:21.653+09:00 INFO 11904 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.8]
2023-05-23T16:10:21.714+09:00 INFO 11904 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-05-23T16:10:21.715+09:00 INFO 11904 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1002 ms
2023-05-23T16:10:21.832+09:00 INFO 11904 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : MyHikariCP - Starting...
2023-05-23T16:10:21.836+09:00 WARN 11904 --- [ restartedMain] c.zaxxer.hikari.util.DriverDataSource : Registered driver with driverClassName=oracle.jdbc.driver.OracleDriver was
2023-05-23T16:10:22.099+09:00 INFO 11904 --- [ restartedMain] com.zaxxer.hikari.pool.HikariPool : MyHikariCP - Added connection oracle.jdbc.driver.T4CConnection@1b627dc5
2023-05-23T16:10:22.101+09:00 INFO 11904 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : MyHikariCP - Start completed.
```

## 2-2. @Configuration 클래스 파일을 이용한 방법

**\*\* src/main/resources/config.properties 파일 새로 만들어서 수행**

**\* config.properties를 따로 만드는 이유 \***

- 1) DB, 이메일 인증 등 계정, 비밀번호가 github에 업로드 되는 문제
- 2) 서버 경로, DB URL, DB 계정 정보 변경 등  
팀원들 간의 서로 다른 정보를 기입하는 상황이 발생하여 코드의 충돌이 발생하는 문데

이러한 문제를 해결하고자 별도의 properties 파일을 만들어서 각자 필요한 정보를 작성하고 ignore 처리를 해서 공유되지 않게 함.

### 1) 오라클 DB와 연결 하는 객체 설정 (config.properties)

```
#오라클 DB 연결 정보
spring.datasource.hikari.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.hikari.jdbc-url=jdbc:oracle:thin:@IP주소:포트번호:DB이름
```

```

spring.datasource.hikari.username=계정명
spring.datasource.hikari.password=비밀번호

# HikariCP Connection Pool Properties
#풀이 연결을 사용할 수 있을 때까지 대기하는 최대 시간(ms)을 지정
spring.datasource.hikari.connection-timeout=30000

#풀의 최대 연결 수 설정
spring.datasource.hikari.maximum-pool-size=5

#연결이 풀에서 유휴 상태로 있을 수 있는 최대 시간(ms)을 지정
spring.datasource.hikari.idle-timeout=600000

#연결 풀의 이름을 지정
spring.datasource.hikari.pool-name=MyHikariCP

#자동 커밋 끄기
spring.datasource.hikari.auto-commit=false

```

## 2) edu.kh.project.common.config.DBConfig 클래스 작성

```

package edu.kh.project.common.config;

import javax.sql.DataSource;

import org.apache.ibatis.session.SqlSessionFactory;
import org.mybatis.spring.SqlSessionFactoryBean;
import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;

import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

@Configuration
// @PropertySource : properties 파일의 내용을 이용하겠다는 어노테이션
// 다른 properties도 추가하고 싶으면 어노테이션을 계속 추가
@PropertySource("classpath:/config.properties")
public class DBConfig{

    @Autowired
    private ApplicationContext applicationContext;

    // @Bean
    // - 개발자가 수동으로 bean을 등록하는 어노테이션
    // - @Bean 어노테이션이 작성된 메서드에서 반환된 객체는
    // Spring Container가 관리함(IOC)

    @Bean
    // @ConfigurationProperties(prefix = "spring.datasource.hikari")

```

```
// properties 파일의 내용을 이용해서 생성되는 bean을 설정하는 어노테이션
// prefix를 지정하여 spring.datasource.hikari으로 시작하는 설정을 모두 적용
@ConfigurationProperties(prefix = "spring.datasource.hikari")
public HikariConfig hikariConfig() {
    return new HikariConfig();
}

@Bean
public DataSource dataSource(HikariConfig config) {
    DataSource dataSource = new HikariDataSource(config);
    return dataSource;
}

}
```

### 3) 애플리케이션을 실행시켜 콘솔 확인



```

:: Spring Boot :: (v3.1.0)

2023-05-23T16:10:20.668+09:00 INFO 11904 --- [ restartedMain] e.k.a.AdminprojectApplication : Starting AdminprojectApplication using Java 17.0.7 with PID 11904 (D:\work
2023-05-23T16:10:20.670+09:00 INFO 11904 --- [ restartedMain] e.k.a.AdminprojectApplication : No active profile set, falling back to 1 default profile: "default"
2023-05-23T16:10:20.712+09:00 INFO 11904 --- [ restartedMain] o.s.b.devtools.restart.ChangeableUrls : The Class-Path manifest attribute in C:\Users\knbdh\gradle\caches\modules
2023-05-23T16:10:20.712+09:00 INFO 11904 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : Devtools property defaults active! Set 'spring.devtools.add-properties' to
2023-05-23T16:10:20.712+09:00 INFO 11904 --- [ restartedMain] .e.DevToolsPropertyDefaultsPostProcessor : For additional web related logging consider setting the 'logging.level.web
2023-05-23T16:10:21.184+09:00 INFO 11904 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2023-05-23T16:10:21.195+09:00 INFO 11904 --- [ restartedMain] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 4 ms. Found 0 JPA repository in
2023-05-23T16:10:21.283+09:00 WARN 11904 --- [ restartedMain] o.m.s.mapper.ClassPathMapperScanner : No MyBatis mapper was found in '[edu.kh.adminproject]' package. Please che
2023-05-23T16:10:21.643+09:00 INFO 11904 --- [ restartedMain] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 80 (http)
2023-05-23T16:10:21.652+09:00 INFO 11904 --- [ restartedMain] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2023-05-23T16:10:21.653+09:00 INFO 11904 --- [ restartedMain] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.8]
2023-05-23T16:10:21.714+09:00 INFO 11904 --- [ restartedMain] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2023-05-23T16:10:21.715+09:00 INFO 11904 --- [ restartedMain] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1002 ms
2023-05-23T16:10:21.832+09:00 INFO 11904 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : MyHikariCP - Starting...
2023-05-23T16:10:21.836+09:00 WARN 11904 --- [ restartedMain] c.zaxxer.hikari.util.DriverDataSource : Registered driver with driverClassName=oracle.jdbc.driver.OracleDriver was
2023-05-23T16:10:22.099+09:00 INFO 11904 --- [ restartedMain] com.zaxxer.hikari.pool.HikariPool : MyHikariCP - Added connection oracle.jdbc.driver.T4CConnection@1b627dc5
2023-05-23T16:10:22.101+09:00 INFO 11904 --- [ restartedMain] com.zaxxer.hikari.HikariDataSource : MyHikariCP - Start completed.
```

## 3. 마이바티스 추가 하기

build.gradle → dependencies에 추가

```
implementation 'org.mybatis.spring.boot:mybatis-spring-boot-starter:3.0.2'
```

- 마이바티스 설정 파일 (mybatis-config.xml) 생성
  - src/main/resources 폴더에 mybatis-config.xml 파일 생성
  - 아래 코드 작성

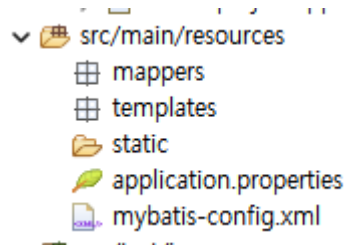
```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE configuration PUBLIC "-//mybatis.org//DTD Config 3.0/EN" "http://mybatis.org/dtd/mybatis-3-config.dtd">
```

```
<configuration>

    <settings>
        <setting name="jdbcTypeForNull" value="NULL" />
        <setting name="mapUnderscoreToCamelCase" value="true" />
    </settings>

</configuration>
```

- src/main/resources 폴더에 mappers 폴더 추가



### 3-1. application.properties 설정 방법

(DB 연결 시 2-1 application.properties를 이용한 연결 방법에서만 사용 가능)

#### application.properties 작성

```
### 마이바티스 설정 ###

#마이바티스 설정 파일 경로
mybatis.config-location=classpath:mybatis-config.xml

#매퍼 파일 경로
mybatis.mapper-locations=classpath:/mappers/**/*.xml

#별칭을 지정할 파일이 포함된 패키지
#coma(,) 구분하여 여러 패키지 작성, 별칭은 클래스명의 소문자로 자동 지정
mybatis.type-aliases-package=edu.kh.project.member.model.dto, edu.kh.project.board.model.dto
```

### 3-2. @Configuration 클래스 파일을 이용한 방법

(DB 연결 시 2-2 @Configuration 클래스 파일을 이용한 연결 방법에서만 사용 가능)

#### edu.kh.project.common.config.DBConfig 클래스

```
package edu.kh.project.common.config;

import javax.sql.DataSource;
```



```

import org.apache.ibatis.session.SqlSessionFactory;
import org.mybatis.spring.SqlSessionFactoryBean;
import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.context.properties.ConfigurationProperties;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.context.annotation.PropertySource;
import org.springframework.jdbc.datasource.DataSourceTransactionManager;

import com.zaxxer.hikari.HikariConfig;
import com.zaxxer.hikari.HikariDataSource;

@Configuration
// @PropertySource : properties 파일의 내용을 이용하겠다는 어노테이션
// 다른 properties도 추가하고 싶으면 어노테이션을 계속 추가
@PropertySource("classpath:/config.properties")
public class DBConfig{

    @Autowired
    private ApplicationContext applicationContext;

    //@Bean
    // - 개발자가 수동으로 bean을 등록하는 어노테이션
    // - @Bean 어노테이션이 작성된 메서드에서 반환된 객체는
    // Spring Container가 관리함(IOC)

    @Bean
    // @ConfigurationProperties(prefix = "spring.datasource.hikari")
    // properties 파일의 내용을 이용해서 생성되는 bean을 설정하는 어노테이션
    // prefix를 지정하여 spring.datasource.hikari으로 시작하는 설정을 모두 적용
    @ConfigurationProperties(prefix = "spring.datasource.hikari")
    public HikariConfig hikariConfig() {
        return new HikariConfig();
    }

    @Bean
    public DataSource dataSource(HikariConfig config) {
        DataSource dataSource = new HikariDataSource(config);
        return dataSource;
    }

    /////////////////////////////////////////////////// Mybatis 설정 추가 ////////////////////////////////////////////

    // SqlSessionFactory : SqlSession을 만드는 객체
    @Bean
    public SqlSessionFactory sessionFactory(DataSource dataSource) throws Exception{
        SqlSessionFactoryBean sessionFactoryBean = new SqlSessionFactoryBean();

        sessionFactoryBean.setDataSource(dataSource);

        // 매퍼 파일이 모여있는 경로 지정
        sessionFactoryBean.setMapperLocations(applicationContext.getResources("classpath:/mapper

```

```

        // 별칭을 지정해야하는 DTO가 모여있는 패키지 지정
        // -> 해당 패키지에 있는 모든 클래스가 클래스명으로 별칭이 지정됨
        sessionFactoryBean.setTypeAliasesPackage("edu.kh.project.member.model.dto");

        // 마이바티스 설정 파일 경로 지정
        sessionFactoryBean.setConfigLocation(applicationContext.getResource("classpath:mybatis-c

        // SqlSession 객체 반환
        return sessionFactoryBean.getObject();
    }

    // SqlSessionTemplate : 기본 SQL 실행 + 트랜잭션 처리
    @Bean
    public SqlSessionTemplate sqlSessionTemplate(SqlSessionFactory sessionFactory) {
        return new SqlSessionTemplate(sessionFactory);
    }

    // DataSourceTransactionManager : 트랜잭션 매니저
    @Bean
    public DataSourceTransactionManager dataSourceTransactionManager(DataSource dataSource) {
        return new DataSourceTransactionManager(dataSource);
    }
}

```

## 4. @Mapper 어노테이션

**@Mapper** 어노테이션은 MyBatis에서 제공하는 어노테이션 중 하나로  
해당 어노테이션이 작성된 인터페이스가 mapper.xml의  
<mapper namespace="패키지명.인터페이스명"> 에 작성된 경우  
두 파일이 연결되어 인터페이스 메서드명으로 mapper.xml의 sql을 호출할 수 있다.

### edu.kh.project.member.model.dao.MemberMapper.jav

```

package edu.kh.project.member.model.dao;

import org.apache.ibatis.annotations.Mapper;

import edu.kh.project.member.model.dto.Member;

// @Mapper : 마이바티스 mapper와 연결된 인터페이스임을 명시
// - 해당 인터페이스에 메서드명과 mapper.xml파일의 id가 일치하는 태그가 자동으로 연결됨

// -> 단, mapper.xml파일의 namespace가 해당 인터페이스의 패키지명+클래스명으로 등록되어 있어야함.

// - @Mapper가 작성된 인터페이스를 사용하면 sqlSessionTemplate 객체를 DI 하지않아도됨(내부 코드 수행 시 자동 주입)

```



```

@Mapper
public interface MemberMapper {

    Member login(Member inputMember);

    int signUp(Member inputMember);
}

```

## src/main/resources/mappers/member-mapper.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-r

<!-- namespace를 Mapper 인터페이스로 지정!! -->
<mapper namespace="edu.kh.project.member.model.mapper.MemberMapper">

    <!-- 로그인 -->
    <select id="login" parameterType="Member" resultType="Member">
        SELECT MEMBER_NO, MEMBER_EMAIL, MEMBER_NICKNAME , MEMBER_PW,
            MEMBER_TEL , MEMBER_ADDRESS , PROFILE_IMG , AUTHORITY ,
            TO_CHAR(ENROLL_DATE, 'YYYY"년" MM"월" DD"일" HH24"시" MI"분" SS"초"') AS ENROLL_DATE
        FROM "MEMBER"
        WHERE MEMBER_DEL_FL = 'N'
        AND MEMBER_EMAIL = #{memberEmail}
    </select>

    <!-- 회원 가입 -->
    <insert id="signup" parameterType="Member" >
        INSERT INTO "MEMBER"
        VALUES(SEQ_MEMBER_NO.NEXTVAL, #{memberEmail}, #{memberPw},
            #{memberNickname}, #{memberTel},
            #{memberAddress},
            NULL, DEFAULT, DEFAULT, DEFAULT
        )
    </insert>

</mapper>

```