Spring Day 14

JDBC + MyBatis

▼ MyBatis 환경설정

pom.xml 설정

```
<!-- https://mvnrepository.com/artifact/org.mybatis/mybatis -->
  <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis</artifactId>
      <version>3.5.6</version>
  </dependency>
      <!-- https://mvnrepository.com/artifact/org.mybatis/mybatis-spring -->
  <dependency>
      <groupId>org.mybatis</groupId>
      <artifactId>mybatis-spring</artifactId>
      <version>2.0.4</version>
  </dependency>
      <!-- https://mvnrepository.com/artifact/org.springframework/spring-jdbc -->
  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-jdbc</artifactId>
      <version>5.3.19</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-dbcp2 -->
  <dependency>
      <groupId>org.apache.commons</groupId>
      <artifactId>commons-dbcp2</artifactId>
      <version>2.7.0</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.bgee.log4jdbc-log4j2/log4jdbc-log4j2-jdbc4 -->
  <dependency>
      <groupId>org.bgee.log4jdbc-log4j2</groupId>
      <artifactId>log4jdbc-log4j2-jdbc4</artifactId>
      <version>1.16</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
  <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.28</version>
  </dependency>
```

servlet-context.xml 설정

```
<!-- 업로드 패스 설정 -->
  <beans:bean class="java.lang.String" id="uploadPath">
    <beans:constructor-arg
      value="C:\develop\JSP\CarShop\src\main\webapp\resources" /> <!-- 로컬에서 작업할 때 -->
  </beans:bean>
```

root-context.xml 설정

```
<!-- mysql 연결 설정 -->
    <bean id="dataSource" class="org.apache.commons.dbcp2.BasicDataSource" destroy-method="close">
    <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
    <property name="url" value="jdbc:mysql://localhost:3306/difbfl4751?serverTimezone=UTC"/>
    <property name="username" value="difbfl4751"/>
    <property name="password" value="kdy3529216"/>
    </bean>
```

```
      <bean id="sqlSessionFactory"
       class="org.mybatis.spring.SqlSessionFactoryBean">
       <property name="dataSource" ref="dataSource" />
       <property name="mapperLocations"
        value="classpath:/sqlmap/**/*_SQL.xml" />
      </bean>
      <bean id="sqlSessionTemplate"
       class="org.mybatis.spring.SqlSessionTemplate">
       <constructor-arg index="0" ref="sqlSessionFactory" />
      </bean>
```

파일경로에서 * 과 ** 의 차이

파일경로 설정시 *, ** 차이

코드를 읽다보니, **/*.js 처럼 파일경로에 *, ** 이 사용되는 경우가 있는데, 두 개의 차이점은 무엇인지 느낌상 ** 은 하위 경로를 모두 선택하는 것 같지만 확실하게 정리해보고자 합니다.

https://velog.io/@rimo09/파일경로-설정시-차이

velog

car_SQL.xml

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

<insert id="insert" parameterType="com.carshop.controller.CarDTO" useGeneratedKeys="true" keyProperty="cid" >
<![CDATA[
  insert into car
  (cid, cname, cprice, ccate, cdesc, cfilename)
  values
  (#{cid}, #{cname}, #{cprice}, #{ccate}, #{cdesc}, #{cfilename})
  ]]>
</insert>

<select id="select_detail" parameterType="String" resultType="com.carshop.controller.CarDTO">
<![CDATA[
  select * from car where cid = #{carid}
  ]]>
</select>

<update id="update" parameterType="com.carshop.controller.CarDTO">
UPDATE car SET cname=#{cname}, cprice=#{cprice}, ccate=#{ccate}, cdesc=#{cdesc}, cfilename=IFNULL(#{cfilename}, cfilename) WHERE cid =
</update>

<delete id="delete" parameterType="String">
    DELETE FROM car WHERE cid = #{carId}
</delete>

<select id="select_list" resultType="com.carshop.controller.CarDTO">
<![CDATA[
  select * from car ORDER BY cprice DESC
  ]]>
</select>

</mapper>
```

JDBC 비활성화

```
package com.carshop.controller;

import java.util.*;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

@Repository
```

```
public class CarRepositoryImpl {
//  public class CarRepositoryImpl implements CarRepository {

  private List<CarDTO> listOfCars = new ArrayList<CarDTO>();

  public List<CarDTO> getCarListByCategory(String category) {
    List<CarDTO> carsByCategory = new ArrayList<CarDTO>();
    for (int i = 0; i < listOfCars.size(); i++) {
      CarDTO carDTO = listOfCars.get(i);
      if (category.equalsIgnoreCase(carDTO.getCcate())) {
        carsByCategory.add(carDTO);
      }
    }
    return carsByCategory;
  }

  private JdbcTemplate template;

  @Autowired
  public void setJDBCTemplate(DataSource dataSource) {
    this.template = new JdbcTemplate(dataSource);
  }


//  @Override
  public List<CarDTO> getAllCarList() {
    String sql = "SELECT * FROM car";
    List<CarDTO> listOfCars = template.query(sql, new CarRowMapper());
    return listOfCars;
  }

//  @Override
  public CarDTO getCarById(String carId) {

    CarDTO carInfo = null;

    String sql = "SELECT count(*) FROM car where cid=?";
    int rowCount = template.queryForObject(sql, Integer.class, carId);

    if(rowCount != 0) {
      sql = "SELECT * FROM car where cid=?";
      carInfo = template.queryForObject(sql, new Object[] {carId}, new CarRowMapper());
    }


    if(carInfo == null) {
      throw new IllegalArgumentException("자동차 ID 가 " + carId + "인 자동차는 없습니다.");
    }


    return carInfo;
  }

//  @Override
  public void setNewCar(CarDTO car) {
    String sql = "INSERT INTO car (cid, cname, cprice, ccate, cdesc, cfilename) "
        + "VALUE (?,?,?,?,?,?)";

    template.update(sql, car.getCid(),
               car.getCname(),
               car.getCprice(),
               car.getCcate(),
               car.getCdesc(),
               car.getCfilename());
  }

//  @Override
  public void deleteCar(String carId) {
    String sql = "DELETE FROM car WHERE cid=?";

    template.update(sql, carId);
  }


//  @Override
  public void setUpdateCar(CarDTO car) {

    String sql = "UPDATE car SET cname=?, cprice=?, ccate=?, cdesc=?, cfilename=IFNULL(?, cfilename) where cid=?";

    template.update(sql, car.getCname(),
                  car.getCprice(),
                  car.getCcate(),
                  car.getCdesc(),
                  car.getCfilename(),
                  car.getCid());

  }
```

```
}
```

JDBC 활성화

```
package com.carshop.controller;

import java.util.*;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

@Repository
//public class CarRepositoryImpl {
  public class CarRepositoryImpl implements CarRepository {

  private List<CarDTO> listOfCars = new ArrayList<CarDTO>();

  public List<CarDTO> getCarListByCategory(String category) {
    List<CarDTO> carsByCategory = new ArrayList<CarDTO>();
    for (int i = 0; i < listOfCars.size(); i++) {
      CarDTO carDTO = listOfCars.get(i);
      if (category.equalsIgnoreCase(carDTO.getCcate())) {
        carsByCategory.add(carDTO);
      }
    }
    return carsByCategory;
  }

  private JdbcTemplate template;

  @Autowired
  public void setJDBCTemplate(DataSource dataSource) {
    this.template = new JdbcTemplate(dataSource);
  }


  @Override
  public List<CarDTO> getAllCarList() {
    String sql = "SELECT * FROM car";
    List<CarDTO> listOfCars = template.query(sql, new CarRowMapper());
    return listOfCars;
  }

  @Override
  public CarDTO getCarById(String carId) {

    CarDTO carInfo = null;

    String sql = "SELECT count(*) FROM car where cid=?";
    int rowCount = template.queryForObject(sql, Integer.class, carId);

    if(rowCount != 0) {
      sql = "SELECT * FROM car where cid=?";
      carInfo = template.queryForObject(sql, new Object[] {carId}, new CarRowMapper());
    }


    if(carInfo == null) {
      throw new IllegalArgumentException("자동차 ID 가 " + carId + "인 자동차는 없습니다.");
    }


    return carInfo;
  }

  @Override
  public void setNewCar(CarDTO car) {
    String sql = "INSERT INTO car (cid, cname, cprice, ccate, cdesc, cfilename) "
        + "VALUE (?,?,?,?,?,?)";

    template.update(sql, car.getCid(),
               car.getCname(),
               car.getCprice(),
               car.getCcate(),
               car.getCdesc(),
               car.getCfilename());
  }
```

```
  @Override
  public void deleteCar(String carId) {
    String sql = "DELETE FROM car WHERE cid=?";

    template.update(sql, carId);
  }


  @Override
  public void setUpdateCar(CarDTO car) {

    String sql = "UPDATE car SET cname=?, cprice=?, ccate=?, cdesc=?, cfilename=IFNULL(?, cfilename) where cid=?";

    template.update(sql, car.getCname(),
                  car.getCprice(),
                  car.getCcate(),
                  car.getCdesc(),
                  car.getCfilename(),
                  car.getCid());

  }


}
```

MyBatisRepositoryImpl.java(MyBatis 비활성화)

```
package com.carshop.controller;

import java.util.List;

import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class MyBatisRepositoryImpl {
//    public class MyBatisRepositoryImpl implements CarRepository {
  @Autowired
  SqlSessionTemplate sqlSessionTemplate;

  //@Override
  public List<CarDTO> getAllCarList() {
    return this.sqlSessionTemplate.selectList("car.select_list");
  }

  @Override
  public List<CarDTO> getCarListByCategory(String category) {
    return null;
  }

  //@Override
  public CarDTO getCarById(String carId) {
    return this.sqlSessionTemplate.selectOne("car.select_detail", carId);
  }

  //@Override
  public void setNewCar(CarDTO car) {
    this.sqlSessionTemplate.insert("car.insert", car);
  }

  //@Override
  public void deleteCar(String carId) {
    this.sqlSessionTemplate.delete("car.delete", carId);
  }

  //@Override
  public void setUpdateCar(CarDTO car) {
    this.sqlSessionTemplate.update("car.update", car);
  }

}
```

MyBatisRepositoryImpl.java(MyBatis 활성화)

```java
package com.carshop.controller;

import java.util.List;

import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class MyBatisRepositoryImpl implements CarRepository {

  @Autowired
  SqlSessionTemplate sqlSessionTemplate;

  @Override
  public List<CarDTO> getAllCarList() {
    return this.sqlSessionTemplate.selectList("car.select_list");
  }

  @Override
  public List<CarDTO> getCarListByCategory(String category) {
    return null;
  }

  @Override
  public CarDTO getCarById(String carId) {
    return this.sqlSessionTemplate.selectOne("car.select_detail", carId);
  }

  @Override
  public void setNewCar(CarDTO car) {
    this.sqlSessionTemplate.insert("car.insert", car);
  }

  @Override
  public void deleteCar(String carId) {
    this.sqlSessionTemplate.delete("car.delete", carId);
  }

  @Override
  public void setUpdateCar(CarDTO car) {
    this.sqlSessionTemplate.update("car.update", car);
  }

}
```

💡 **이와 같은 수동 방식으로 인터페이스를 구현하면 헷갈리기 쉽고 번거롭다.**
이를 해결하기 위해서 @Primary 어노테이션을 사용할 수 있다.

@Primary

```java
//해당 Repository에 우선순위를 지정한다.
@Primary
@Repository
public class MyBatisRepositoryImpl implements CarRepository {
```

```java
@Repository
public class CarRepositoryImpl implements CarRepository {
```

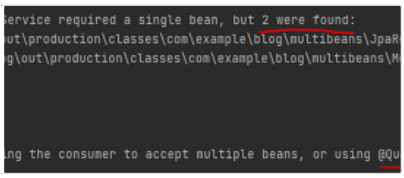
💡 **이와 같은 방식을 사용하면 2개의 Bean을 활성화시켜 놓을 수 있다.**

여러개의 빈(Bean)을 구현시키는 방법

[Spring] 조회 빈(Bean)이 2개 이상일때 처리하는 방법

스프링 컨테이너에 빈을 등록하면 스프링 컨테이너가 알아서 의존관계를 맺어준다. 그런데 의존관계를 맺어줄 때 해당하는 타입의 빈이 2개 이상이라면 어떤 문제가 발생할까? 간단한 예시로 알아보도록 하자. 현재 MyRepository 인터페이스가 하나 있으며, 이를 구현하는 2개의 Repository가 있다. public interface MyRepository { } @Repository

https://sorjfkrh5078.tistory.com/273



[Spring] 같은 타입의 빈이 여러개일 때 해결방법

프로젝트 진행할 때 "DB는 안 정해졌는데 언젠가 정해질것이니 개발을 하시오." 이런 상황을 겪어봤을 것이다..Spring을 입문자라 이럴때면 주석처리를 해서 해결을 하는 나도 속터지고 너도 속터지는 방법을 사용해 개발을 했었다..😡 이를 해결하는 방법을 알아보자.

https://velog.io/@kero1004/Spring-같은타입의빈이여러개일때해결방법

Spring Security

▼ DB생성

💡 **Spring Security 는 기본적인 DB연동 서비스를 제공한다.**

💡 **DB연동을 위해서는 테이블 이름(users, authorities)과 컬럼 이름 (username, password, authority)을 Spring Security의 지정된 변수명과 통일시켜야 한다.**

users

이름: users

코멘트:

열: 추가 제거 위로 아래로

#	이름	데이터 유형	길이/설정	부호 없...	NULL 허...	0으로...	기본값	코멘트	조합	표현식
1	username	VARCHAR	50	☐	☐	☐	기본값 없음		utf8mb4_0900_ai_ci	
2	password	VARCHAR	100	☐	☐	☐	기본값 없음		utf8mb4_0900_ai_ci	
3	enabled	TINYINT	1	☐	☐	☐	'0'			

```
CREATE TABLE `users` (
  `username` VARCHAR(50) NOT NULL COLLATE 'utf8mb4_0900_ai_ci',
  `password` VARCHAR(100) NOT NULL COLLATE 'utf8mb4_0900_ai_ci',
  `enabled` TINYINT(1) NOT NULL DEFAULT '0'
)
COLLATE='utf8mb4_0900_ai_ci'
```

```
ENGINE=InnoDB
;
```

authorities

기본 | 옵션 | 인덱스 (1) | 외래 키 (0) | 제약 조건 확인 (0) | 분할 | CREATE 코드 | ALTER 코드

이름: authorities

코멘트:

열: 추가 제거 위로 아래로

#	이름	데이터 유형	길이/설정	부호 없...	NULL 허...	0으로...	기본값	코멘트	조합	표현식
1	username	VARCHAR	50	☐	☑	☐	NULL		utf8mb4_0900_ai_ci	
2	authority	VARCHAR	50	☐	☑	☐	NULL		utf8mb4_0900_ai_ci	

```
CREATE TABLE `authorities` (
  `username` VARCHAR(50) NULL DEFAULT NULL COLLATE 'utf8mb4_0900_ai_ci',
  `authority` VARCHAR(50) NULL DEFAULT NULL COLLATE 'utf8mb4_0900_ai_ci',
  UNIQUE INDEX `ix_auth_username` (`username`, `authority`) USING BTREE
)
COLLATE='utf8mb4_0900_ai_ci'
ENGINE=InnoDB
;
```

예시 데이터

difbfl4751.authorities: 4 행 (총) (대략적)

username	authority
admin	ROLE_ADMIN
admin	USER
admin	USER_MANAGER
kim	USER

difbfl4751.users: 2 행 (총) (대략적)

username	password	enabled
kim	$2a$10$lbdTX16tpVENpDSKwws1h.wBPco2ZYE...	1
admin	$2a$10$lbdTX16tpVENpDSKwws1h.wBPco2ZYE...	1

```
INSERT INTO `users` (`username`, `password`, `enabled`) VALUES
  ('kim', '$2a$10$lbdTX16tpVENpDSKwws1h.wBPco2ZYEHsshWz5S44N.f8W/f1Hmja', 1),
  ('admin', '$2a$10$lbdTX16tpVENpDSKwws1h.wBPco2ZYEHsshWz5S44N.f8W/f1Hmja', 1);

INSERT INTO `authorities` (`username`, `authority`) VALUES
  ('admin', 'ROLE_ADMIN'),
  ('admin', 'USER'),
  ('admin', 'USER_MANAGER'),
  ('kim', 'USER');
```

security-context.xml 설정

```
<?xml version="1.0" encoding="UTF-8"?>
<beans
  xmlns="http://www.springframework.org/schema/beans"
  xmlns:security="http://www.springframework.org/schema/security"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans.xsd
  http://www.springframework.org/schema/security
  http://www.springframework.org/schema/security/spring-security.xsd">


<!-- 각각의  interceptr-url, form-login, logout 은  내부적으로 Filter를 만들어 사용한다. -->
<!-- 그래서 web.xml에서 이 모든걸 엮어줄 FilterChaing을 따로 설정해준다. -->
<!-- web.xml에서 사용하는 FilterChain의 대한 설정부분이다. -->
  <security:http use-expressions="true">
<!--  <http auto-config="true" use-expressions="true"> 기본 로그인 창 사용시 -->
  <security:intercept-url pattern="/cars/add/**" access="hasAuthority('ROLE_ADMIN')" /> <!-- ADMIN만 허용 -->
  <security:intercept-url pattern="/manager" access="hasRole('ROLE_MANAGER')" /> <!-- MANAGER만 허용 -->
  <security:intercept-url pattern="/member" access="isAuthenticated()" /> <!-- 로그인한 사람만 허용 -->
  <security:intercept-url pattern="/**" access="permitAll" /> <!-- 전체 허용 -->

  <security:form-login login-page="/cars/login"
      default-target-url="/"
      authentication-failure-url="/cars/loginfailed"
      username-parameter="username"
      password-parameter="password"/>

  <security:csrf/>

  <security:logout logout-success-url="/cars/logout"/>

  </security:http>

<!--  <authentication-manager> -->
<!--    <authentication-provider> -->
<!--    <user-service> -->
<!--  <user name="admin" password="{noop}admin" authorities="ROLE_ADMIN, ROLE_USER" /> -->
<!--  <user name="manager" password="{noop}manager" authorities="ROLE_MANAGER, ROLE_USER" /> -->
<!--  <user name="guest" password="{noop}guest" authorities="ROLE_USER" /> -->
<!--    </user-service> -->
<!--    </authentication-provider> -->
<!--  </authentication-manager> -->

  <!-- 암호화를 위한 passwordEncoder -->
<bean id="passwordEncoder" class="org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder"></bean>

  <!-- DB연동은 data-source만 지정해주면 된다 -->
  <security:authentication-manager alias="authenticationManager">
    <security:authentication-provider>
     <security:jdbc-user-service data-source-ref="dataSource" />
        <security:password-encoder ref="passwordEncoder"></security:password-encoder>
    </security:authentication-provider>
  </security:authentication-manager>


  <!-- 이것만 있으면  JDBC 코드 (Connection, Statement,ResultSet)로   DB연결 가능 -->
  <bean id="dataSource"
    class="org.apache.commons.dbcp2.BasicDataSource"
    destroy-method="close">
    <property name="driverClassName"
      value="com.mysql.cj.jdbc.Driver" />
    <property name="url" value="jdbc:mysql://localhost:3306/difbfl4751?characterEncoding=utf8" />
    <property name="username" value="difbfl4751" />
    <property name="password" value="kdy3529216" />
    <property name="defaultAutoCommit" value="true" />
  </bean>

</beans>
```

UsersController.java

```
package com.carshop.users;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PostMapping;

@Controller
public class UsersController {
```

```
  @Autowired
  UserService userService;

  @GetMapping("/join")
  public String joinForm(@ModelAttribute("NewUser") User user) {
    return "users/joinform";
  }

  @PostMapping("/join")
  public String submitForm(@ModelAttribute("NewUser") User user) {
    userService.setNewUser(user);
    return "redirect:/cars/login";
  }

}
```

UserService.java

```
package com.carshop.users;

public interface UserService {

  void setNewUser(User user);

}
```

UserServiceImpl.java

```
package com.carshop.users;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

@Service
public class UserServiceImpl implements UserService {

  @Autowired
  UserRepository userRepository;

  @Override
  public void setNewUser(User user) {

    userRepository.setNewUser(user);

  }

}
```

UserRepository.java

```
package com.carshop.users;

public interface UserRepository {

  void setNewUser (User user);
}
```

UserRepositoryImpl.java

```
package com.carshop.users;

import org.mybatis.spring.SqlSessionTemplate;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;

@Repository
public class UserRepositoryImpl implements UserRepository {
```

```
  @Autowired
  SqlSessionTemplate sqlSessionTemplate;

  @Override
  public void setNewUser(User user) {
    this.sqlSessionTemplate.insert("user.insert", user);
  }

}
```

user_SQL.xml

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

<!-- insert : MyBatis의 데이터 입력 태그 -->
<!-- id : insert 태그의 id -->
<!-- parameterType : 데이터의 형태 -->
<!-- useGeneratedKeys & keyProperty : useGeneratedKeys를 true로 설정하면 MyBatis에서 insert 쿼리 실행 후 생성된 PK를 keyProperty 속성에 넣어준다.
<insert id="insert" parameterType="com.carshop.users.User" useGeneratedKeys="true" keyProperty="username" >
<![CDATA[
  insert into users
  (username, password, enabled)
  values
  (#{username}, #{password}, 1)
  ]]>
</insert>

</mapper>
```

joinform.jsp

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form" %>

<head>
  <title>자동차 등록</title>
  <%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
      <meta charset="utf-8">

</head>
<body class="text-center">
<div class="alert alert-dark" role="alert">
<div class="container"><h1>차량 등록</h1>

      <form:form modelAttribute="NewUser" action="./join?${_csrf.parameterName}=${_csrf.token}" class="form-horizontal" enctype="multi
        <fieldset>
        <legend>
        ${addTitle }
        </legend>
          username : <form:input path="username" class="form-control" />
          password : <form:input path="password" class="form-control" />

        <input type="submit" class="btn btn-primary" value="등록">


        </fieldset>
      </form:form>





    </div>
</div>
```

```
</body>
</html>
```

INT의 종류 및 크기

int의 종류 및 크기 tinyint, smallint, int, bigint 초간단 정리

int가 들어가는 데이터 타입의 크기가 가끔 헷갈리는데 간단하게 정리합니다. int는 inteager의 약자입니다. int는 정수입니다. 소숫점을 사용하고 싶은 분은 decimal 같은 데이터 형식을 사용하세요 :) #크기비교 tinyint < smallint < int < bigint #tinyint 크기 : 0 ~ 255 비고 : 0을 시작으로 2^8(=2의8승=256)번째까지 정수 용량 : 1바이트 #smallint 크기 :

https://heavening.tistory.com/85