

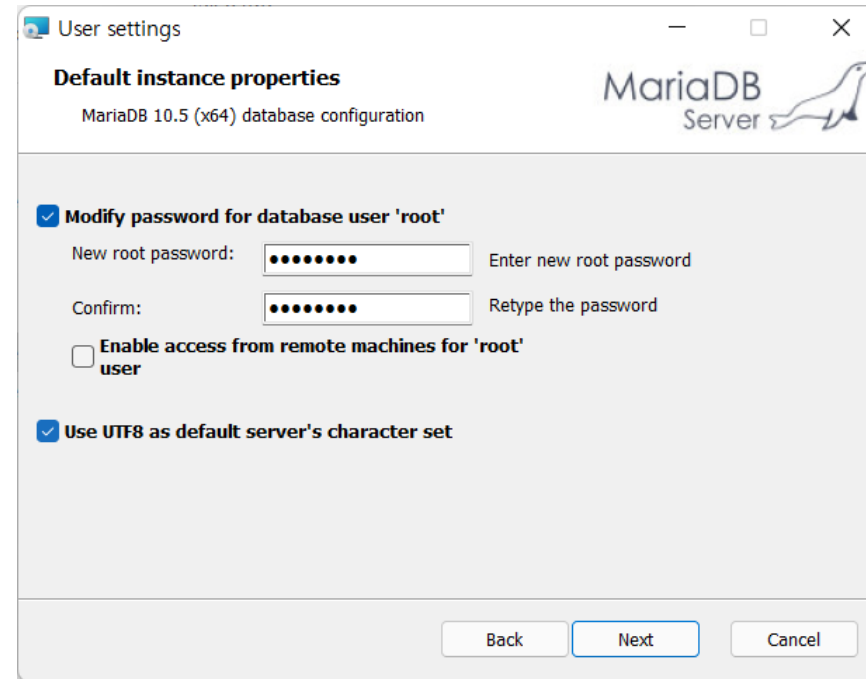
# PART2 -웹과 데이터베이스

# JDBC 프로그래밍의 준비

- JDBC(Java Database Connectivity)는 자바에서 데이터베이스에 접속할 수 있도록 하는 자바 API
- JDBC 프로그래밍 - Java를 이용해서 데이터베이스와 연동해 원하는 작업을 처리하는 프로그램을 작성

# MariaDB 설치와 생성

- MariaDB의 다운로드(<https://mariadb.org/>)
- MySQL과 거의 동일한 기능을 제공하지만 무료 사용이 가능
- SQL 편집 도구
  - HeidiSQL 기본 제공
  - MySQL의 Workbench
  - DataGrip(Intellij)
- 설치 과정시 주의할 점
  - UTF-8을 이용하도록 설정
  - ROOT 계정에 대한 패스워드 기억 필요



User settings

Default instance properties  
MariaDB 10.5 (x64) database configuration

MariaDB Server

☒ **Modify password for database user 'root'**

New root password:  Enter new root password

Confirm:  Retype the password

☐ **Enable access from remote machines for 'root' user**

☒ **Use UTF8 as default server's character set**

Back Next Cancel

# 데이터베이스 생성과 사용자 생성

- ROOT계정으로 접속한 후 데이터베이스 생성

- `CREATE DATABASE webdb;`

- 사용자 계정 생성

- 접근할 수 있는 IP 지정 가능

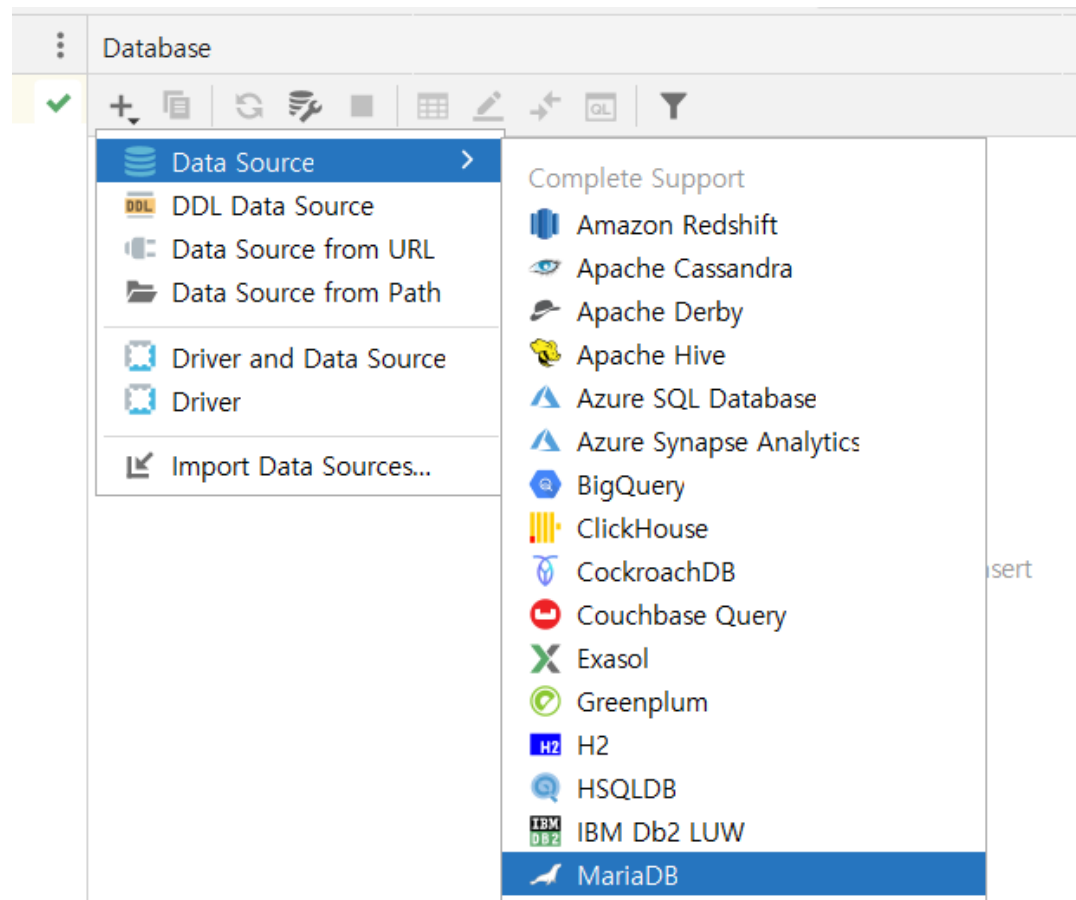
- `CREATE USER `webuser`@`%` IDENTIFIED BY `webuser`;`

- 계정내 권한 부여

- `GRANT ALL PRIVILEGES ON webdb.* TO `webuser`@`%`;`

# IntelliJ의 MariaDB설정

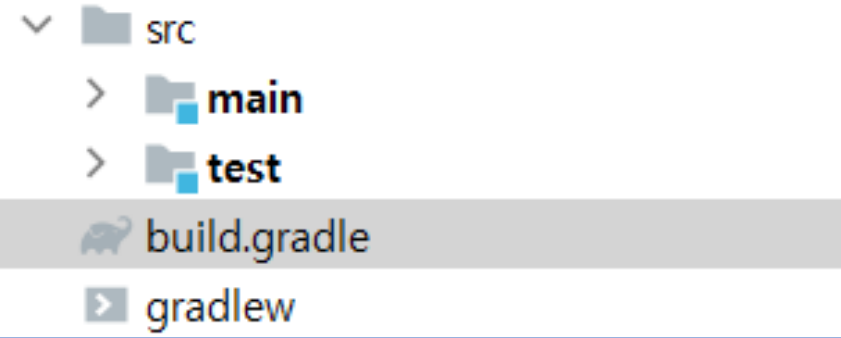
- IntelliJ Ultimate의 경우 Database메뉴를 이용해서 현재 프로젝트에서 사용하는 데이터베이스를 설정해서 사용할 있음



# 프로젝트내 MariaDB 연동

- JDBC를 이용하는 경우 네트워크 통신에 대한 처리를 위한 JDBC Driver 필요

The screenshot shows the Maven Repository page for `org.mariadb.jdbc`. The page includes a search bar, a graph of indexed artifacts (26.1M), and a list of popular categories. The main content area displays the **MariaDB Java Client** with its license (LGPL 2.1), categories (MySQL Drivers), tags (database, sql, jdbc, driver, client, mysql), and a table of versions (3.0.3, 3.0.2-rc, 3.0.1-beta) with their respective vulnerabilities.



```
dependencies {
    compileOnly('javax.servlet:javax.servlet-api:4.0.1')

    testImplementation("org.junit.jupiter:junit-jupiter-api:${junitVersion}")
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junitVersion}")

    implementation 'org.mariadb.jdbc:mariadb-java-client:3.0.3'
}
```

# JDBC 프로그램의 구조/작성 순서

- JDBC 프로그램의 데이터베이스 처리는 JDBC 드라이버에서 제공하는 API를 이용하는 구조
- API들은 모두 `java.sql` 혹은 `javax.sql`의 인터페이스 구현



1. 네트워크를 통해서 데이터베이스와 연결을 맺는 단계
2. 데이터베이스에 보낼 SQL을 작성하고 전송하는 단계
3. 필요하다면 데이터베이스가 보낸 결과를 받아서 처리하는 단계
4. 데이터베이스와 연결을 종료하는 단계

# JDBC 연결 테스트

- Junit을 이용해서 데이터베이스 연결이 가능한지 테스트

```
package org.zerock.dao;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;

import java.sql.Connection;
import java.sql.DriverManager;

public class ConnectTests {

    @Test
    public void testConnection() throws Exception {

        Class.forName("org.mariadb.jdbc.Driver");

        Connection connection = DriverManager.getConnection(
            "jdbc:mariadb://localhost:3306/webdb",
            "webuser",
            "webuser");

        Assertions.assertNotNull(connection);

        connection.close();
    }
}
```



- `Class.forName( )`: JDBC드라이버 클래스를 메모리 상으로 로딩하는 역할을 합니다. 이 때 문자열은 패키지명과 클래스명의 대소문자까지 정확히 일치해야 합니다. 만일 JDBC 드라이버 파일이 없는 경우에는 이 부분에서 예외가 발생하게 됩니다.
- `Connection connection`: `java.sql`의 `Connection` 인터페이스 타입의 변수입니다. `Connection`은 데이터베이스와 연결이 정상적으로 이루어졌을때만 생성됩니다.
- `DriverManager.getConnection( )`: 데이터베이스내에 있는 여러 정보들을 통해서 특정한 데이터베이스(예제에서는 `webdb`)에 연결을 시도합니다.
  - '`jdbc:mariadb://localhost:3306/webdb`' 는 `jdbc`프로토콜이라는 것을 이용한다는 의미이고, `localhost:3306`은 네트워크 연결 정보를, `webdb`는 연결하려는 데이터베이스 정보를 의미합니다.
  - `webuser`: 연결을 위해서는 사용자의 계정과 패스워드가 필요합니다.
- `Assertions.assertNotNull( )`: 데이터베이스와 정상적으로 연결이 된다면 `Connection` 타입의 객체는 `null`이 아니라는 것을 확신한다는 의미입니다.
- `connection.close( )`: 데이터베이스와의 연결을 종료합니다. JDBC 프로그램은 데이터베이스와 연결을 잠깐씩 맺고 종료하는 방식으로 처리됩니다. 따라서 반드시 작업이 완료되면 데이터베이스와의 연결을 종료해주어야만 합니다.

# 데이터베이스내 테이블 생성

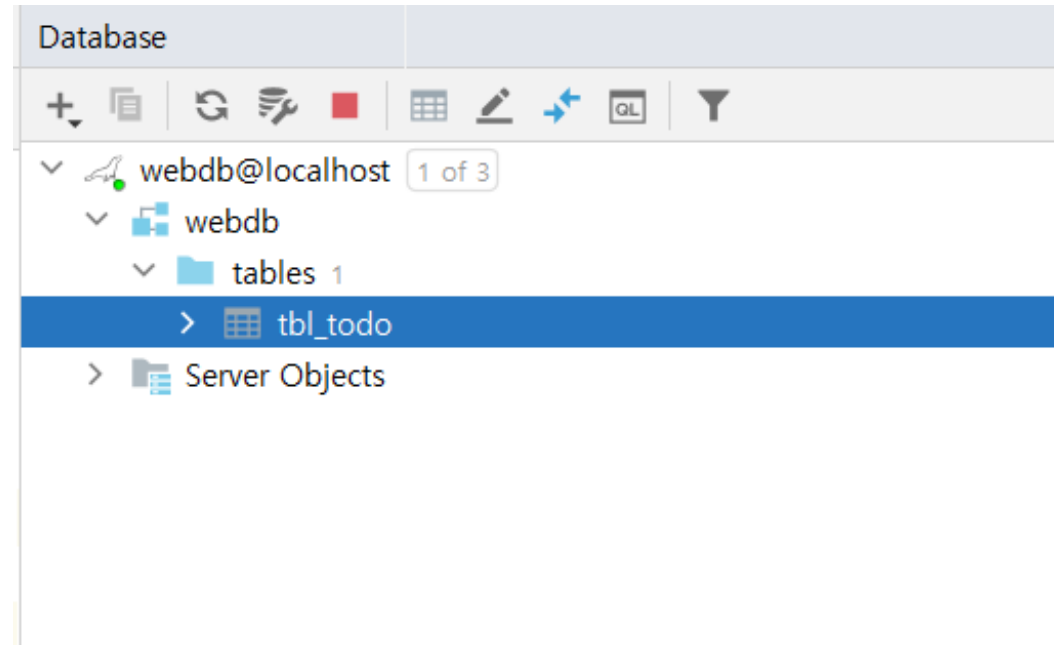
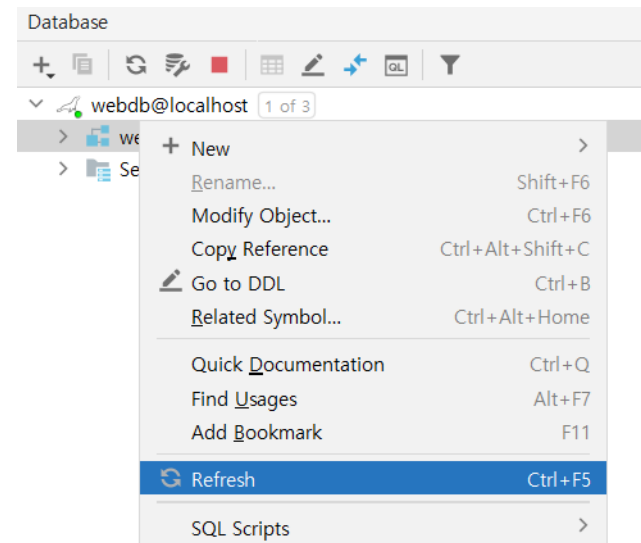
## • MaiaDB의 타입

타입	용도	크기	설명
TINYINT	매우 작은 정수	1 byte	-128 ~ 127 ( 0 ~ 255)
SMALLINT	작은 정수	2 byte	-32768 ~ 32767(0 ~65535)
MEDIUMINT	중간 크기의 정수	3 byte	-8388608 ~ 8388607(0~16777215)
INT	표준 정수	4 byte	-2147483648 ~ 2147483647 (0 ~ 4294967295)
BIGINT	큰 정수	8 byte	-9223372036854775808 ~ 9223372036854775807 (0 ~ 18446744073709551615)
FLOAT	단정도 부동 소수	4 byte	-3.40E+45 ~ 3.40E+45 (no unsigned)
DOUBLE	배정도 부동 소수	8 byte	-1.7976E+320 ~ 1.7976E+320 (no unsigned)
DECIMAL(m,n)	고정 소수	m과 n에 따라 다르다	숫자 데이터지만 내부적으로 String형태로 저장됨. 최대 65자.
BIT(n)	비트 필드	m에 따라 다르다	

데이터 타입	형태	크기	설명
DATE	CCYY-MM-DD	3 byte	1000-01-01 ~ 9999-12-31
DATETIME	CCYY-MM-DD hh:mm:ss	8 byte	1000-01-01 00:00:00 ~ 9999-12-31 23:59:59
TIMESTAMP	CCYY-MM-DD hh:mm:ss	4 byte	1970-01-01 00:00:00 ~ 2037
TIME	hh:mm:ss	3 byte	-839:59:59 ~ 839:59:59
YEAR	CCYY 또는 YY	1 byte	1901 ~ 2155

데이터 타입	용도	크기	설명
CHAR(n)	고정길이 비이진(문자) 문자열	n byte	
VARCHAR(n)	가변 길이 비이진 문자열	Length + 1 byte	
BINARY(n)	고정길이 이진 문자열	n byte	
VARBINARY(n)	가변 길이 이진 문자열	Length + 1 byte or 2 byte	
TINYBLOB	매우작은 BLOB(Binary Large Object)	Length + 1 byte	
BLOB	작은 BLOB	Length + 2 byte	최대크기 64KB
MEDIUMBLOB,	중간 크기 BLOB	Length + 3 byte	최대크기 16MB
LOBLOB	큰 BLOB	Length + 4 byte	최대크기 4GB
TINYTEXT	매우 작은 비이진 문자열	Length + 1 byte	
TEXT	작은 비이진 문자열	Length + 2 byte	최대크기 64KB
MEDIUMTEXT	중간 크기 비이진 문자열	Length + 3 byte	최대크기 16MB
LONGTEXT	큰 비이진 문자열	Length + 4 byte	최대크기 4GB

```
1 ✓ create table tbl_todo (
2     tno int auto_increment primary key ,
3     title varchar(100) not null,
4     dueDate date not null,
5     finished tinyint default 0
6 );
```



```
CREATE TABLE webdb.tbl_todo (
    tno INT AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(100) NOT NULL,
    dueDate DATE NOT NULL,
    finished TINYINT DEFAULT 0,
    writer VARCHAR(100)
);
```

# 데이터의 추가

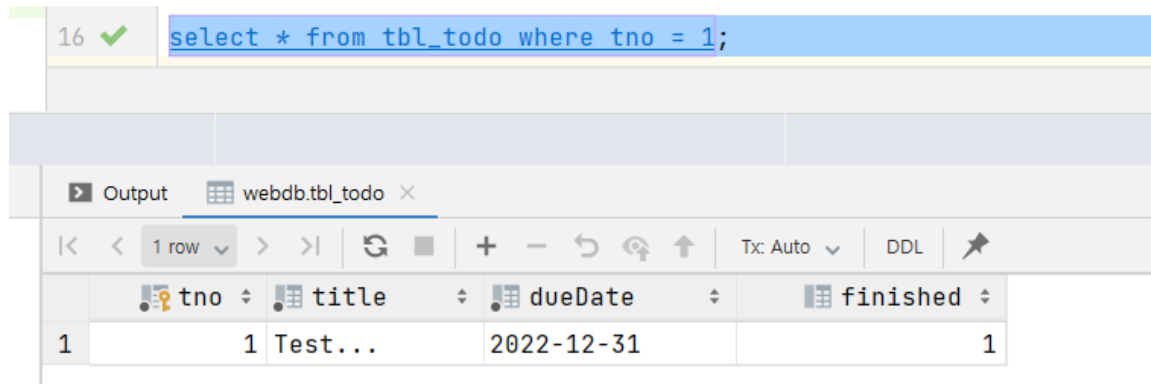
- `insert into tbl_todo (title, dueDate, finished)`  
`values ('Test...', '2022-12-31', 1);`
- auto\_increment 값은 자동으로 증가

```
select * from tbl_todo;
```

Output webdb.tbl_todo			
5 rows > >  ↺ ■ + - ↶ ↷ ↕ Tx: Auto DDL ↗			
tno	title	dueDate	finished
1	Test...	2022-12-31	1
2	Test...	2022-12-31	1
3	Test...	2022-12-31	1
4	Test...	2022-12-31	1
5	Test...	2022-12-31	1

# 데이터-select

- 데이터를 조회하는 SQL – 쿼리(query)
- `select * from tbl_todo where tno = 1`



The screenshot shows a database client interface. At the top, a SQL query is entered in a text area: `select * from tbl_todo where tno = 1;`. Below the query area, there is a tab labeled "webdb.tbl\_todo". Underneath the tab is a toolbar with various icons for navigation and execution. Below the toolbar is a table displaying the results of the query. The table has four columns: `tno`, `title`, `dueDate`, and `finished`. The first row of data shows `tno` as 1, `title` as "Test...", `dueDate` as "2022-12-31", and `finished` as 1.

tno	title	dueDate	finished
1	Test...	2022-12-31	1

# 데이터 – update/delete

- update 문은 set 을 이용해서 특정한 컬럼의 내용을 수정할 수 있고, where 조건을 이용해서 수정하는 대상 데이터들을 지정
- delete 문은 where 조건에 해당하는 데이터들을 삭제하기 때문에 주의

# DML과 쿼리의 차이

- DML(Data Manipulation Language)의 경우
  - insert/update/delete
  - 결과는 몇 건이 영향을 받았는지에 대해서 반환
- 쿼리(select)의 경우
  - 데이터의 결과 집합(Result Set)을 반환
  - 한번에 모든 데이터를 반환하거나 하나씩 반환하는 방식이 아닌
  - 일정한 양의 데이터를 모아서 반환
  - 필요한 경위 추가적으로 데이터를 반환

# JDBC를 위한 API와 용어들

- java.sql.Connection

- Connection인터페이스는 데이터베이스와의 네트워크 상의 연결을 의미
- JDBC 프로그래밍에서 가장 중요한 사실은 'Connection은 반드시 close( )해야 한다'
- Connection의 종료를 위해서는 코드 내에서 try~catch~finally를 이용해서 종료하거나 try-with-resources 방식을 이용

- java.sql.Statement/PreparedStatement

- SQL을 데이터베이스로 보내기 위해서는 Statement/PreparedStatement 타입을 이용
- SQL Injection 공격방지를 위해서는 PreparedStatement만 사용

- java.sql.ResultSet

- 쿼리(select)의 실행 결과를 외부 프로그램에서 얻을 수 있는 스트림
- next( )를 이용해서 커서(cursor)를 이동시켜가면서 결과 추출



# close( )의 의미

- Connection/Statement/ResultSet 은 반드시 close( )
- 데이터베이스 입장에서는 JDBC 처리가 끝났는지 알 수가 없으므로 사용한 뒤에 close( )를 해 주어야만 데이터베이스내의 자원들을 해제
- close( )를 하는 방법
  - try ~ catch ~ finally
  - try with resources
  - Lombok의 @Cleanup

# Connection Pool과 DataSource

- 네트워크를 통한 Connection의 연결은 많은 시간과 리소스의 소가 필요함
- 미리 커넥션을 연결해두고 보관하면서 필요할때마다 사용후 반반하는 개념의 Connection Pooling
- javax.sql.DataSource는 Connection Pool을 인터페이스로 정리
- Connection Pool 라이브러리
  - DBCP
  - C3PO
  - HikariCP

# DAO(Data Access Object)

- 데이터베이스와의 연동을 전문적으로 처리하는 객체
- 비즈니스 로직과 무관하도록 구성하고 데이터베이스에 원하는 작업을 처리하는 메소드들로 구성
- 파라미터나 리턴타입은 DTO혹은 VO
- VO(Value Object) – 순수한 데이터 자체를 의미하는 객체로 주로 getter로만 구성하는 읽기 전용 객체

# 프로젝트내 JDBC 구현

# Lombok 라이브러리

- 코드 작성시 자주 사용하는 코드들을 컴파일 시점에 생성해 주는 라이브러리
- getter/setter/생성자/toString( )/equals( )...
- 어노테이션을 이용
  - 프로젝트의 라이브러리로 추가되어야 하고, 개발 도구에서 지원 필요
  - 이클립스의 경우 IDE 자체에 Lombok관련 설정 추가 필요

# Lombok라이브러리 추가

Home » org.projectlombok » lombok » 1.18.22

**Project Lombok » 1.18.22**

Spice up your java: Automatic Resource Management, automatic generation of getters, setters, equals, hashCode and toString, and more!

License	MIT
HomePage	<a href="https://projectlombok.org">https://projectlombok.org</a>
Date	(Oct 07, 2021)
Files	<a href="#">pom (1 KB)</a> <a href="#">jar (1.9 MB)</a> <a href="#">View All</a>
Repositories	Central
Used By	13,821 artifacts

Maven Gradle Gradle (Short) Gradle (Kotlin) SBT Ivy Grape Leiningen Buildr

```
// https://mvnrepository.com/artifact/org.projectlombok/lombok
compileOnly group: 'org.projectlombok', name: 'lombok', version: '1.18.22'
```

```
dependencies {
    compileOnly('javax.servlet:javax.servlet-api:4.0.1')

    testImplementation("org.junit.jupiter:junit-jupiter-api:${junitVersion}")
    testRuntimeOnly("org.junit.jupiter:junit-jupiter-engine:${junitVersion}")

    implementation 'org.mariadb.jdbc:mariadb-java-client:3.0.3'

    compileOnly group: 'org.projectlombok', name: 'lombok', version: '1.18.22'

    annotationProcessor group: 'org.projectlombok', name: 'lombok', version: '1.18.22'
}
```

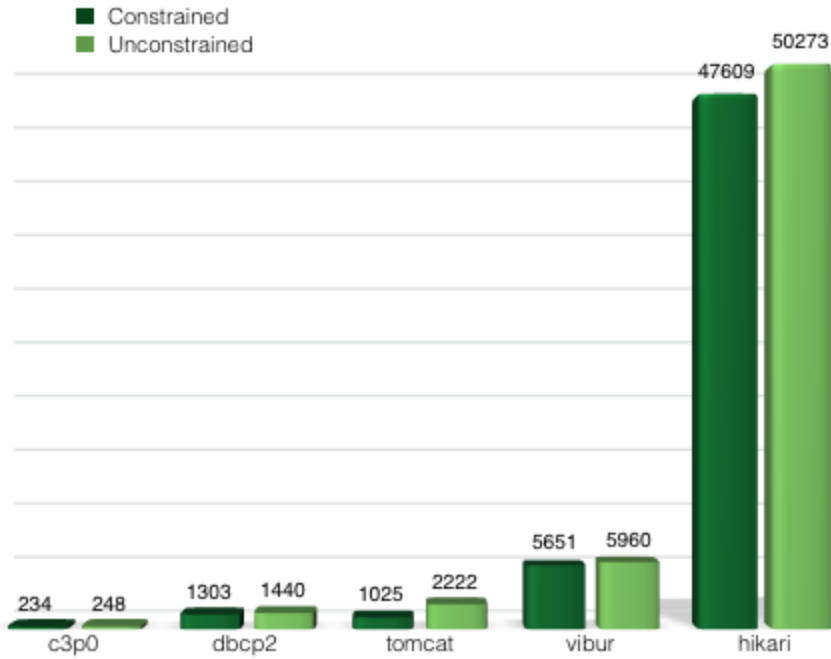
IntelliJ의 경우 test환경에서 사용하기 위해서는 testCompileOnly와 testAnnotationProcessor 가 필요함

# HikariCP의 설정

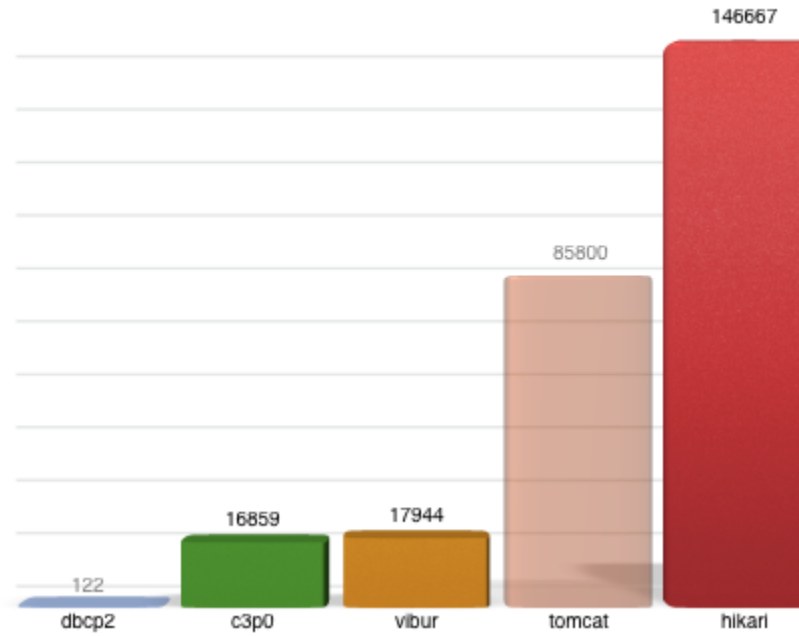
- <https://github.com/brettwooldridge/HikariCP>

```
dependencies {  
    compileOnly('javax.servlet:javax.servlet-api:4.0.1')  
  
    ...생략...  
  
    implementation group: 'com.zaxxer', name: 'HikariCP', version: '5.0.0'  
}
```

Connection Cycle ops/ms



Statement Cycle ops/ms



# ConnectionUtil

- DAO에서 Connection들을 쉽게 이용할 수 있는 싱글턴 객체로 작성
- 내부적으로 HikariCP를 이용

```
public enum ConnectionUtil {  
  
    INSTANCE;  
  
    private HikariDataSource ds;  
  
    ConnectionUtil() {  
        HikariConfig config = new HikariConfig();  
        config.setDriverClassName("org.mariadb.jdbc.Driver");  
        config.setJdbcUrl("jdbc:mariadb://localhost:3306/webdb");  
        config.setUsername("webuser");  
        config.setPassword("webuser");  
        config.addDataSourceProperty("cachePrepStmts", "true");  
        config.addDataSourceProperty("prepStmtCacheSize", "250");  
        config.addDataSourceProperty("prepStmtCacheSqlLimit", "2048");  
  
        ds = new HikariDataSource(config);  
    }  
  
    public Connection getConnection() throws Exception {  
        return ds.getConnection();  
    }  
  
}
```



# @Cleanup

- close( )처리 코드를 자동으로 컴파일 시점에 완성
- 별도의 try~ 등의 코드가 필요하지 않으므로 간결한 코드 완성
- Lombok 라이브러리에 종속적이라는 단점

# 웹 MVC와 JDBC의 결합

# ModelMapper라이브러리

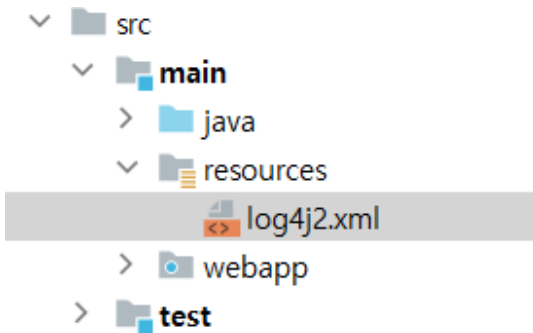
- 비슷한 구조의 객체를 복제하는 작업에 유용
- DTO -> VO 혹은 VO -> DTO처리시 유용

```
public enum MapperUtil {  
    INSTANCE;  
  
    private ModelMapper modelMapper;  
  
    MapperUtil() {  
        this.modelMapper = new ModelMapper();  
        this.modelMapper.getConfiguration()  
            .setFieldMatchingEnabled(true)  
            .setFieldAccessLevel(org.modelmapper.config.Configuration.AccessLevel.PRIVATE)  
            .setMatchingStrategy(MatchingStrategies.LOOSE);  
    }  
  
    public ModelMapper get() {  
        return modelMapper;  
    }  
}
```

# Log4j2와 @Log4j2

- System.out.print... 대신에 Log4j2
- Appender – 로그의 기록 대상
- Level – 로그의 수준
  - Trace > Debug > Info > Warn > Error > Fatal

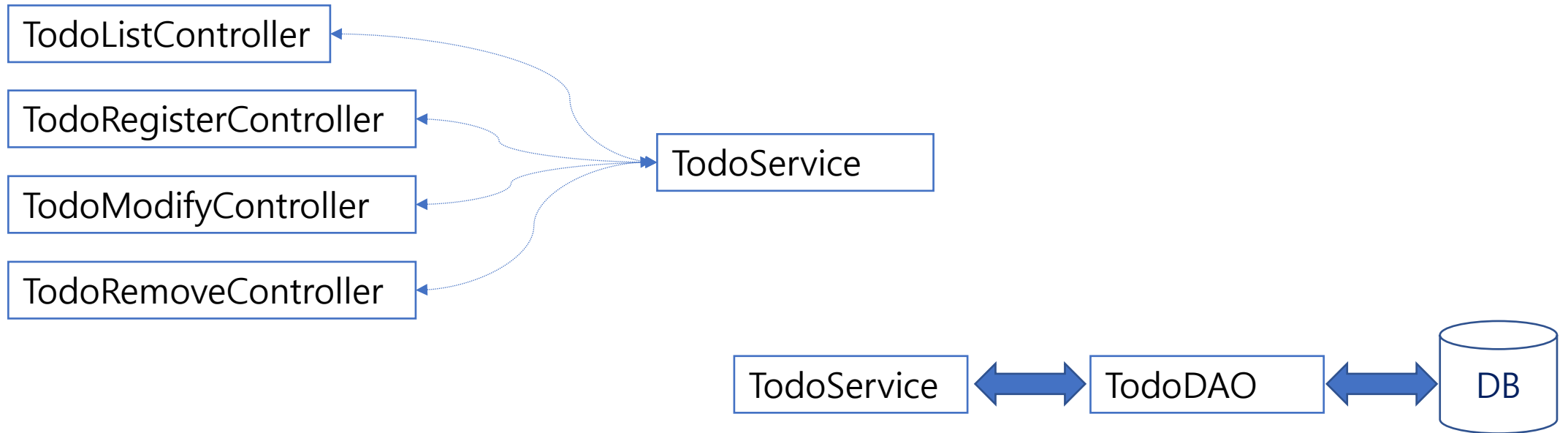
```
implementation group: 'org.apache.logging.log4j', name: 'log4j-core', version: '2.17.2'  
implementation group: 'org.apache.logging.log4j', name: 'log4j-api', version: '2.17.2'  
implementation group: 'org.apache.logging.log4j', name: 'log4j-slf4j-impl', version:
```



```
<?xml version="1.0" encoding="UTF-8"?>  
<Configuration status="WARN">  
  <Appenders>  
    <Console name="Console" target="SYSTEM_OUT">  
      <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level  
%logger{36} - %msg%n"/>  
    </Console>  
  </Appenders>  
  <Loggers>  
    <Root level="info">  
      <AppenderRef ref="Console"/>  
    </Root>  
  </Loggers>  
</Configuration>
```

# 컨트롤러와 서비스 객체의 연동

- 여러 개의 컨트롤러(Servlet)에서 하나의 서비스 객체를 연동할 필요



기능	동작 방식	컨트롤러	JSP
목록	GET	TodoListController	WEB-INF/todo/list.jsp
등록(입력)	GET	TodoRegisterController	WEB-INF/todo/register.jsp
등록(처리)	POST	TodoRegisterController	Redirect
조회	GET	TodoReadController	WEB-INF/todo/read.jsp
수정(입력)	GET	TodoModifyController	WEB-INF/todo/modify.jsp
수정(처리)	POST	TodoModifyController	Redirect
삭제(처리)	POST	TodoRemoveController	Redirect

등록 /todo/register

목록 /todo/list

**Todo List**

- 2 Test... 2021-12-31 DONE
- 3 Not Yet... 2021-12-31 NOT YET
- 4 Test... 2021-12-31 DONE
- 5 Test... 2021-12-31 DONE
- 7 Sample Title... 2021-12-31 NOT YET
- 8 Sample Title... 2021-12-31 NOT YET
- 9 JDBC Test Title 2021-11-28 NOT YET
- 10 JDBC Test Title 2021-11-28 NOT YET
- 11 JDBC Test Title 2021-11-29 NOT YET
- 12 Modified 12 Todo... 2021-12-02 NOT YET

조회 /todo/read?tno=xx

수정삭제 /todo/modify?tno=xx