IN28MINUTES

Integrating Spring Boot and Spring JDBC with H2 and Starter JDBC

This guide will help you create a simple project with Spring Boot. You will add code to the project to connect to a database using Spring JDBC. You will learn to write all the CRUD methods.

You will learn

- How to use Spring Boot Started JDBC?
- How to connect a Spring Boot project to database using Spring JDBC?
- How to write a simple repository class with all the CRUD methods?
- How to execute basic queries using Spring JDBC?
- How to create a project using Spring Boot, Spring JDBC and H2?
- What are the basics of an in memory database?

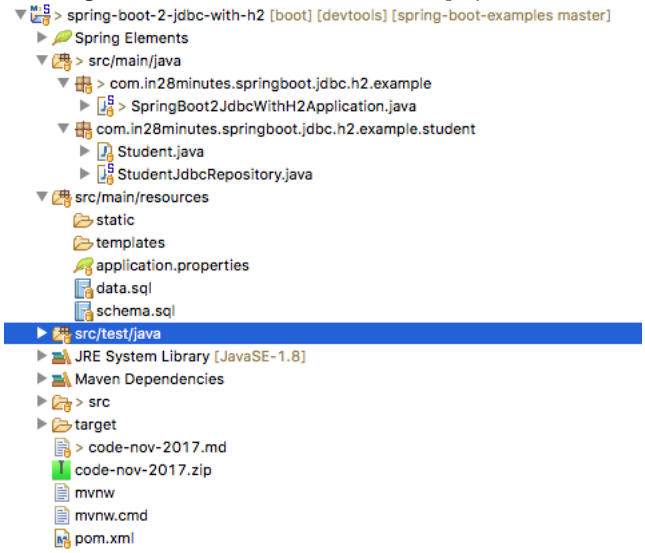
Spring Boot and Hibernate Courses

10 Step Reference Courses

- Spring Framework for Beginners in 10 Steps
- Spring Boot for Beginners in 10 Steps
- Spring MVC in 10 Steps
- JPA and Hibernate in 10 Steps
- Eclipse Tutorial for Beginners in 5 Steps
- Maven Tutorial for Beginners in 5 Steps
- JUnit Tutorial for Beginners in 5 Steps
- Mockito Tutorial for Beginners in 5 Steps
- Complete in28Minutes Course Guide

Project Code Structure

Following screenshot shows the structure of the project we will create.



A few details:

- `Student.java` - The bean to store student details.
- `StudentJdbcRepository.java` - Contains all the methods to store and retrieve student details to the H2 database.
- `schema.sql` - Since we are using an in memory database, we define the tables as part of our application code in this file.
- `data.sql` - We use data.sql to populate the initial student data.
- SpringBoot2JdbcWithH2Application.java - The main Spring Boot Application class which is used to launch up the application. We will extend `CommandLineRunner` interface and implement `public void run(String... args)` method to launch the spring jdbc code when the server launches up.
- `pom.xml` - Contains all the dependencies needed to build this project. We will use Spring Boot Starter JDBC and Web other than Developer Tools and H2 as in memory database.

Tools you will need

- Maven 3.0+ is your build tool
- Your favorite IDE. We use Eclipse.
- JDK 1.8+

Complete Maven Project With Code Examples

> *Our Github repository has all the code examples - https://github.com/in28minutes/spring-boot-examples/tree/master/spring-boot-2-jdbc-with-h2*

A little bit of Theory

JDBC

- JDBC stands for Java Database Connectivity
- It used concepts like Statement, PreparedStatement and ResultSet
- In the example below, the query used is `Update todo set user=?, desc=?, target_date=?, is_done=? where id=?`
- The values needed to execute the query are set into the query using different set methods on the PreparedStatement
- Results from the query are populated into the ResultSet. We had to write code to liquidate the ResultSet into objects.

Update Todo

```
Connection connection = datasource.getConnection();

PreparedStatement st = connection.prepareStatement(
                "Update todo set user=?, desc=?, target_date=?, is_done=? where id=?");

st.setString(1, todo.getUser());
st.setString(2, todo.getDesc());
st.setTimestamp(3, new Timestamp(
                todo.getTargetDate().getTime()));
st.setBoolean(4, todo.isDone());
st.setInt(5, todo.getId());

st.execute();

st.close();

connection.close();
```

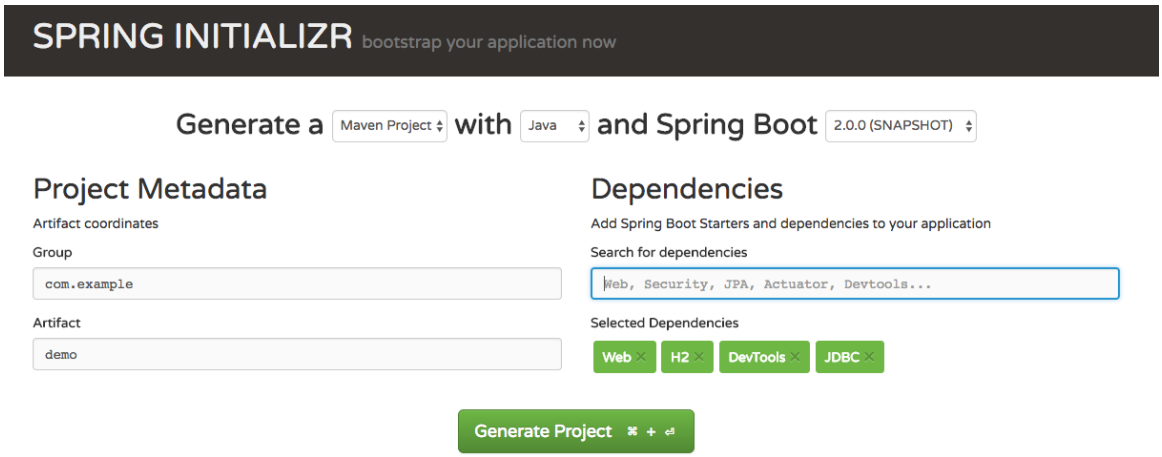
Spring JDBC

- Spring JDBC provides a layer on top of JDBC
- It used concepts like JDBCTemplate
- Typically needs lesser number of lines compared to JDBC as following are simplified
    - mapping parameters to queries
    - liquidating resultsets to beans
    - zero exception handling needed because all exceptions are converted to RuntimeExceptions.

Creating the Project with Spring Initializr

Creating a REST service with Spring Initializr is a cake walk. We will use Spring Web MVC as our web framework.

Spring Initializr http://start.spring.io/ is great tool to bootstrap your Spring Boot projects.



As shown in the image above, following steps have to be done

- Launch Spring Initializr and choose the following
    - Choose `com.in28minutes.springboot.rest.example` as Group
    - Choose `spring-boot-2-jdbc-with-h2` as Artifact
    - Choose following dependencies
        - Web
        - JDBC
        - H2
        - DevTools
- Click Generate Project.
- Import the project into Eclipse. File -> Import -> Existing Maven Project.
- If you want to understand all the files that are part of this project, you can go here TODO.

Starter Projects in pom.xml

Below is the list of starter projects in pom.xml.

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-jdbc</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-devtools</artifactId>
        <scope>runtime</scope>
</dependency>

<dependency>
        <groupId>com.h2database</groupId>
        <artifactId>h2</artifactId>
        <scope>runtime</scope>
</dependency>
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-test</artifactId>
        <scope>test</scope>
</dependency>
```

Initialize H2 in-memory database with the schema

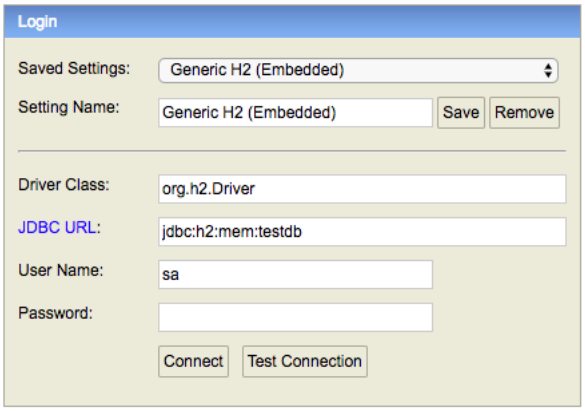
We will use H2 as the database.

H2 provides a web interface called H2 Console to see the data. Let's enable h2 console in the application.properties

/src/main/resources/application.properties

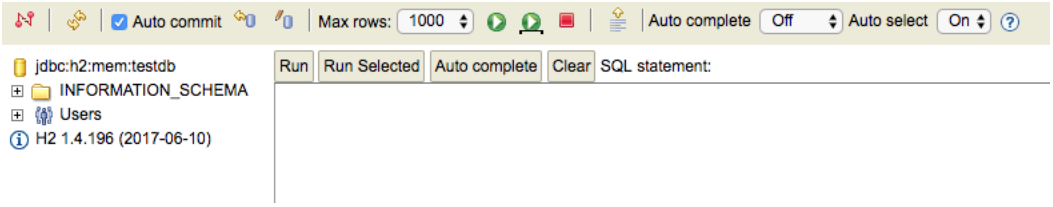
```
# Enabling H2 Console
spring.h2.console.enabled=true
```

When you reload the application, you can launch up H2 Console at http://localhost:8080/h2-console.



Tip - Make sure that you use `jdbc:h2:mem:testdb` as JDBC URL.

When you use the right JDBC URL given above, you should see an empty schema when you click `Connect` button.



Create Schema using schema.sql and Data using data.sql

We will create a table called student with few simple columns. We can initialize a schema by create a `schema.sql` file in the resources.

/src/main/resources/schema.sql

```
create table student
(
   id integer not null,
   name varchar(255) not null,
   passport_number varchar(255) not null,
   primary key(id)
);
```

Let's also populate some data into the student table.

/src/main/resources/data.sql

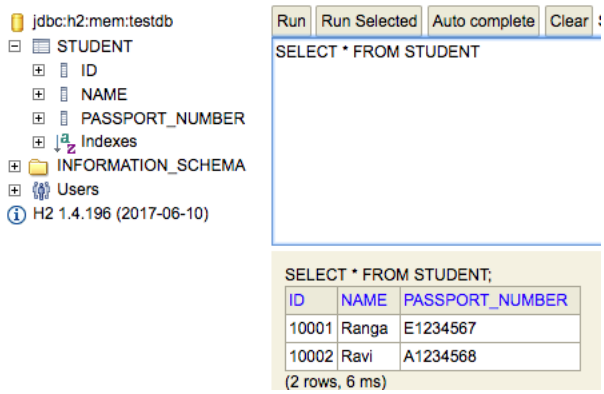
```
insert into student
values(10001,'Ranga', 'E1234567');

insert into student
values(10002,'Ravi', 'A1234568');
```

When the application reloads you would see following statements in the log indicating that the sql files are picked up

```
Executing SQL script from URL [file:/in28Minutes/git/spring-boot-examples/spring-boot-2-jdbc-with-h2/target/cla
Executing SQL script from URL [file:/in28Minutes/git/spring-boot-examples/spring-boot-2-jdbc-with-h2/target/cla
```

When you login to H2 Console (http://localhost:8080/h2-console) you can see that the student table is created and the data is populated.



Creating Student Bean

Lets create a simple Student bean with basic student information along with getters, setters and a toString method.

```
package com.in28minutes.springboot.jdbc.h2.example.student;

public class Student {
	private Long id;
	private String name;
	private String passportNumber;

	public Student() {
		super();
	}

	public Student(Long id, String name, String passportNumber) {
		super();
		this.id = id;
		this.name = name;
		this.passportNumber = passportNumber;
	}

	public Student(String name, String passportNumber) {
		super();
		this.name = name;
		this.passportNumber = passportNumber;
	}

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getPassportNumber() {
		return passportNumber;
	}

	public void setPassportNumber(String passportNumber) {
		this.passportNumber = passportNumber;
	}

	@Override
	public String toString() {
		return String.format("Student [id=%s, name=%s, passportNumber=%s]", id, name, passportNumber);
	}

}
```

Create Repository method to Read Student information

We would want to start with creating a simple repository. To talk to the database we will use a JdbcTemplate.

```
@Repository
public class StudentJdbcRepository {
	@Autowired
	JdbcTemplate jdbcTemplate;
```

> *Spring Boot Auto Configuration sees H2 in the classpath. It understands that we would want to talk to an in memory database. It auto configures a datasource and also a JdbcTemplate connecting to that datasource.*

Let's create the findById method to retrieve a student by id in StudentJdbcRepository.

```
	public Student findById(long id) {
		return jdbcTemplate.queryForObject("select * from student where id=?", new Object[] { id },
				new BeanPropertyRowMapper<Student>(Student.class));
	}
```

Notes

- `JdbcTemplate` has a number of methods to execute queries. In this example, we are using the `queryForObject` method.
- `new Object[] { id }` - We are passing id as a parameter to the query
- `new BeanPropertyRowMapper<Student>(Student.class)` - We are using a BeanPropertyRowMapper to map the results from ResultSet to the Student bean.

We would want to execute findById method. To keep things simple we will make the SpringBoot2JdbcWithH2Application class implement CommandLineRunner and implement run method to call the findById method on the repository.

/src/main/java/com/in28minutes/springboot/jdbc/h2/example/SpringBoot2JdbcWithH2Application.java

```
@SpringBootApplication
public class SpringBoot2JdbcWithH2Application implements CommandLineRunner {

	private Logger logger = LoggerFactory.getLogger(this.getClass());

	@Autowired
	StudentJdbcRepository repository;

	public static void main(String[] args) {
		SpringApplication.run(SpringBoot2JdbcWithH2Application.class, args);
	}

	@Override
	public void run(String... args) throws Exception {

		logger.info("Student id 10001 -> {}", repository.findById(10001L));
	}
}
```

Notes

- `@Autowired StudentJdbcRepository repository;` - We will autowire StudentJdbcRepository we created earlier.
- `public void run(String... args) throws Exception {` - Implement the run method defined in the CommandLineRunner interface. This method is executed as soon as the application is launched up.
- `logger.info("Student id 10001 -> {}", repository.findById(10001L))` - Log all the information about student id 10001.

When the application reloads you will see this in the log.

```
Student id 10001 -> Student [id=10001, name=Ranga, passportNumber=E1234567]
```

> *Congratulations on executing the first Spring JDBC method! You can see how easy it is with Spring Boot.*

Let's now add another method to retrieve details of all the students to StudentJdbcRepository.

```
	class StudentRowMapper implements RowMapper<Student> {
		@Override
		public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
			Student student = new Student();
			student.setId(rs.getLong("id"));
			student.setName(rs.getString("name"));
			student.setPassportNumber(rs.getString("passport_number"));
			return student;
		}

	}

	public List<Student> findAll() {
		return jdbcTemplate.query("select * from student", new StudentRowMapper());
	}
```

Notes

- `class StudentRowMapper implements RowMapper<Student>` - We are defining a custom row mapper to map the result set to student bean.
- `jdbcTemplate.query("select * from student", new StudentRowMapper())` - Since we want to return a list of students we use the `query` method in JdbcTemplate.

We can add a call to find all method in the run method of `SpringBoot2JdbcWithH2Application.java`

```
logger.info("All users 1 -> {}", repository.findAll());
```

You can see following output in the log when the project reloads.

```
All users 1 -> [Student [id=10001, name=Ranga, passportNumber=E1234567], Student [id=10002, name=Ravi, passport
```

Implementing Spring JDBC delete, insert and update methods

The code below shows the delete, insert and update methods.

```
public int deleteById(long id) {
        return jdbcTemplate.update("delete from student where id=?", new Object[] { id });
}

public int insert(Student student) {
        return jdbcTemplate.update("insert into student (id, name, passport_number) " + "values(?,  ?,
                        new Object[] { student.getId(), student.getName(), student.getPassportNumber()
}

public int update(Student student) {
        return jdbcTemplate.update("update student " + " set name = ?, passport_number = ? " + " where
                        new Object[] { student.getName(), student.getPassportNumber(), student.getId()
}
```

Notes

- These methods are relatively straight forward.
- All methods use the `update` method in JdbcTemplate class and set the right query and parameters.

We can add a call to all the above methods in the run method of `SpringBoot2JdbcWithH2Application.java`

```
logger.info("Inserting -> {}", repository.insert(new Student(10010L, "John", "A1234657")));

logger.info("Update 10001 -> {}", repository.update(new Student(10001L, "Name-Updated", "New-Passport")));

repository.deleteById(10002L);

logger.info("All users 2 -> {}", repository.findAll());
```

You can see following output in the log when the project reloads.

```
Inserting -> 1
Update 10001 -> 1
All users 2 -> [Student [id=10001, name=Name-Updated, passportNumber=New-Passport], Student [id=10010, name=Joh
```

> *Awesome ! You've implemented all the CRUD methods using Spring JDBC and Spring Boot.*

> *Congratulations! You are reading an article from a series of 50+ articles on Spring Boot and Microservices. We also have 20+ projects on our Github repository. For the complete series of 50+ articles and code examples, click here.*

Next Steps

Complete Code Example

/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>

	<groupId>com.in28minutes.springboot.rest.example</groupId>
	<artifactId>spring-boot-2-jdbc-with-h2</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>jar</packaging>

	<name>spring-boot-2-jdbc-with-h2</name>
	<description>Spring Boot 2, JDBC and H2 - Example Project</description>

	<parent>
		<groupId>org.springframework.boot</groupId>
		<artifactId>spring-boot-starter-parent</artifactId>
		<version>2.0.0.RELEASE</version>
		<relativePath/> <!-- lookup parent from repository -->
	</parent>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
		<project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>
		<java.version>1.8</java.version>
```

```
	</properties>

	<dependencies>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-jdbc</artifactId>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-web</artifactId>
		</dependency>

		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-devtools</artifactId>
			<scope>runtime</scope>
		</dependency>
		<dependency>
			<groupId>com.h2database</groupId>
			<artifactId>h2</artifactId>
			<scope>runtime</scope>
		</dependency>
		<dependency>
			<groupId>org.springframework.boot</groupId>
			<artifactId>spring-boot-starter-test</artifactId>
			<scope>test</scope>
		</dependency>
	</dependencies>

	<build>
		<plugins>
			<plugin>
				<groupId>org.springframework.boot</groupId>
				<artifactId>spring-boot-maven-plugin</artifactId>
			</plugin>
		</plugins>
	</build>

	<repositories>
		<repository>
			<id>spring-snapshots</id>
			<name>Spring Snapshots</name>
			<url>https://repo.spring.io/snapshot</url>
			<snapshots>
				<enabled>true</enabled>
			</snapshots>
		</repository>
		<repository>
			<id>spring-milestones</id>
			<name>Spring Milestones</name>
			<url>https://repo.spring.io/milestone</url>
			<snapshots>
				<enabled>false</enabled>
			</snapshots>
		</repository>
	</repositories>

	<pluginRepositories>
		<pluginRepository>
			<id>spring-snapshots</id>
			<name>Spring Snapshots</name>
			<url>https://repo.spring.io/snapshot</url>
			<snapshots>
				<enabled>true</enabled>
			</snapshots>
		</pluginRepository>
		<pluginRepository>
			<id>spring-milestones</id>
			<name>Spring Milestones</name>
			<url>https://repo.spring.io/milestone</url>
			<snapshots>
				<enabled>false</enabled>
			</snapshots>
		</pluginRepository>
	</pluginRepositories>


</project>
```

/src/main/java/com/in28minutes/springboot/jdbc/h2/example/SpringBoot2JdbcWithH2A

```
package com.in28minutes.springboot.jdbc.h2.example;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.in28minutes.springboot.jdbc.h2.example.student.Student;
import com.in28minutes.springboot.jdbc.h2.example.student.StudentJdbcRepository;
```

```
@SpringBootApplication
public class SpringBoot2JdbcWithH2Application implements CommandLineRunner {

	private Logger logger = LoggerFactory.getLogger(this.getClass());

	@Autowired
	StudentJdbcRepository repository;

	public static void main(String[] args) {
		SpringApplication.run(SpringBoot2JdbcWithH2Application.class, args);
	}

	@Override
	public void run(String... args) throws Exception {

		logger.info("Student id 10001 -> {}", repository.findById(10001L));

		logger.info("All users 1 -> {}", repository.findAll());

		logger.info("Inserting -> {}", repository.insert(new Student(10010L, "John", "A1234657")));

		logger.info("Update 10001 -> {}", repository.update(new Student(10001L, "Name-Updated", "New-Pa

		repository.deleteById(10002L);

		logger.info("All users 2 -> {}", repository.findAll());

	}
}
```

/src/main/java/com/in28minutes/springboot/jdbc/h2/example/student/Student.java

```
package com.in28minutes.springboot.jdbc.h2.example.student;

public class Student {
	private Long id;
	private String name;
	private String passportNumber;

	public Student() {
		super();
	}

	public Student(Long id, String name, String passportNumber) {
		super();
		this.id = id;
		this.name = name;
		this.passportNumber = passportNumber;
	}

	public Student(String name, String passportNumber) {
		super();
		this.name = name;
		this.passportNumber = passportNumber;
	}

	public Long getId() {
		return id;
	}

	public void setId(Long id) {
		this.id = id;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

	public String getPassportNumber() {
		return passportNumber;
	}

	public void setPassportNumber(String passportNumber) {
		this.passportNumber = passportNumber;
	}

	@Override
	public String toString() {
		return String.format("Student [id=%s, name=%s, passportNumber=%s]", id, name, passportNumber);
	}

}
```

/src/main/java/com/in28minutes/springboot/jdbc/h2/example/student/StudentJdbcRepos

```
package com.in28minutes.springboot.jdbc.h2.example.student;

import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.List;

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

@Repository
public class StudentJdbcRepository {
	@Autowired
	JdbcTemplate jdbcTemplate;

	class StudentRowMapper implements RowMapper<Student> {
		@Override
		public Student mapRow(ResultSet rs, int rowNum) throws SQLException {
			Student student = new Student();
			student.setId(rs.getLong("id"));
			student.setName(rs.getString("name"));
			student.setPassportNumber(rs.getString("passport_number"));
			return student;
		}

	}

	public List<Student> findAll() {
		return jdbcTemplate.query("select * from student", new StudentRowMapper());
	}

	public Student findById(long id) {
		return jdbcTemplate.queryForObject("select * from student where id=?", new Object[] { id },
				new BeanPropertyRowMapper<Student>(Student.class));
	}

	public int deleteById(long id) {
		return jdbcTemplate.update("delete from student where id=?", new Object[] { id });
	}

	public int insert(Student student) {
		return jdbcTemplate.update("insert into student (id, name, passport_number) " + "values(?,  ?,
				new Object[] { student.getId(), student.getName(), student.getPassportNumber()
	}

	public int update(Student student) {
		return jdbcTemplate.update("update student " + " set name = ?, passport_number = ? " + " where
				new Object[] { student.getName(), student.getPassportNumber(), student.getId()
	}

}
```

/src/main/resources/application.properties

```
# Enabling H2 Console
spring.h2.console.enabled=true
#Turn Statistics on
spring.jpa.properties.hibernate.generate_statistics=true
logging.level.org.hibernate.stat=debug
# Show all queries
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
logging.level.org.hibernate.type=trace
```

/src/main/resources/data.sql

```
insert into student
values(10001,'Ranga', 'E1234567');

insert into student
values(10002,'Ravi', 'A1234568');
```

/src/main/resources/schema.sql

```
create table student
(
   id integer not null,
   name varchar(255) not null,
   passport_number varchar(255) not null,
   primary key(id)
);
```

/src/test/java/com/in28minutes/springboot/jdbc/h2/example/SpringBoot2JdbcWithH2Ap

```
package com.in28minutes.springboot.jdbc.h2.example;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.springframework.boot.test.context.SpringBootTest;
import org.springframework.test.context.junit4.SpringRunner;

@RunWith(SpringRunner.class)
@SpringBootTest
public class SpringBoot2JdbcWithH2ApplicationTests {

        @Test
        public void contextLoads() {
        }

}
```
