

Spring Boot In Memory H2 database

springboot04

Spring Boot H2 database

❑ Database in-memory

- Il database in memoria si basa sulla **memoria di sistema** anziché sullo spazio su disco per l'archiviazione dei dati.
- Si usa il database in memoria quando non c'è bisogno di rendere persistenti i dati.
- Il database in memoria è un database incorporato. I database in memoria sono volatili, per impostazione predefinita, e tutti i dati memorizzati perdono quando si riavvia l'applicazione.
- I database in memoria ampiamente utilizzati sono **H2**, **HSQLDB** (HyperSQL Database) e **Apache Derby**. Le configurazioni sono completamente automatiche.

❑ Persistenza vs. in-memory database

- Lo schema e i dati vengono popolati alla partenza dell'applicazione
- I database in-memory sono utili per **POCs** (proof of concepts) e non per applicazioni di produzione
- Il database in-memory più usato è **H2**

Spring Boot H2 database

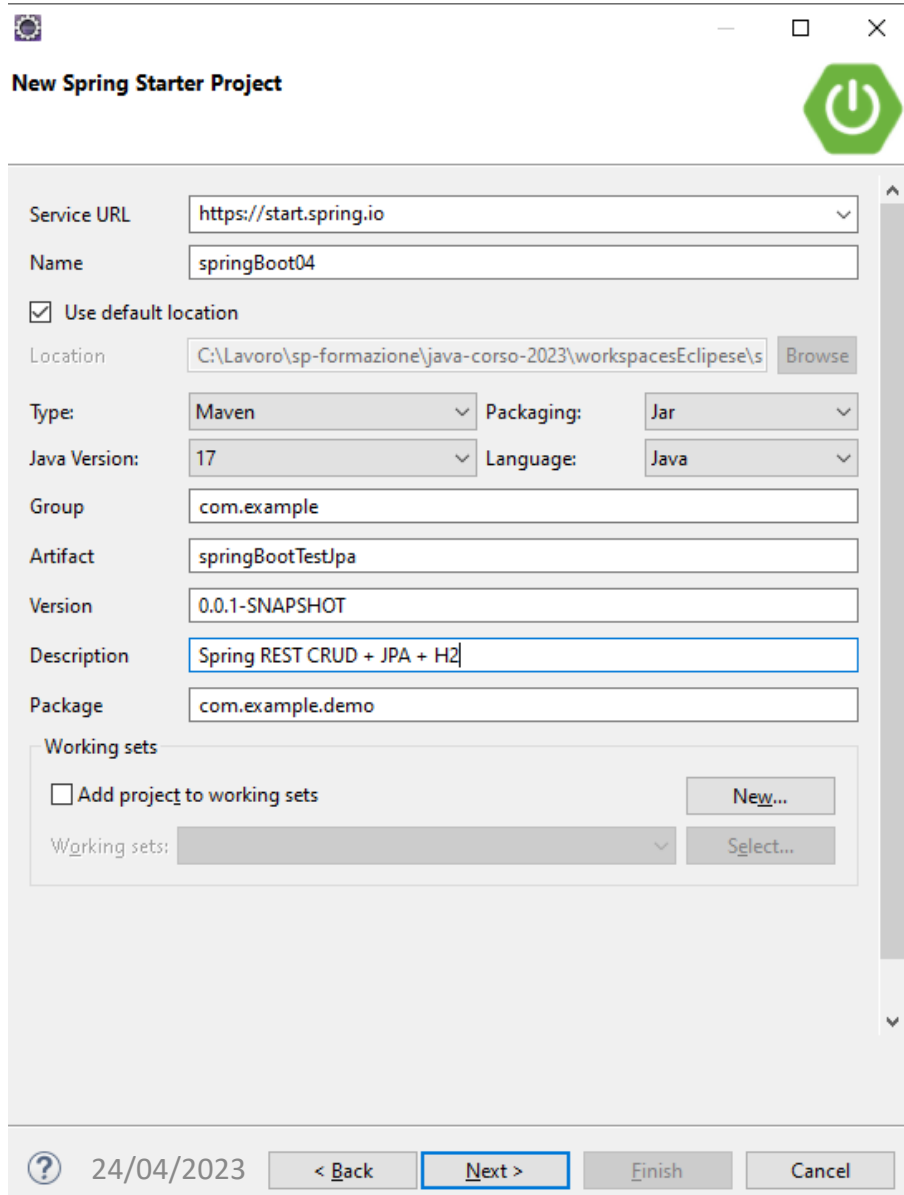
❑ H2 Database

- H2 è un embedded, open-source, e in-memory database .
- E' un dbms relazionale scritto in java ed è una applicazione **client/server**.
- Generalmente usato in unit testing, memorizza i dati in memoria e non persiste su disco.

❑ Vantaggi

- Zero configurazione
- Semplice da utilizzare
- Leggero e veloce
- Semplice configurazione per passare a un dbms reale
- Fornisce una console web per gestire il database
- Viene creato in automatico il database con le tabelle dichiarate via @Entity

Spring Boot H2 database



New Spring Starter Project

Service URL:

Name:

☒ Use default location

Location:

Type: Packaging:

Java Version: Language:

Group:

Artifact:

Version:

Description:

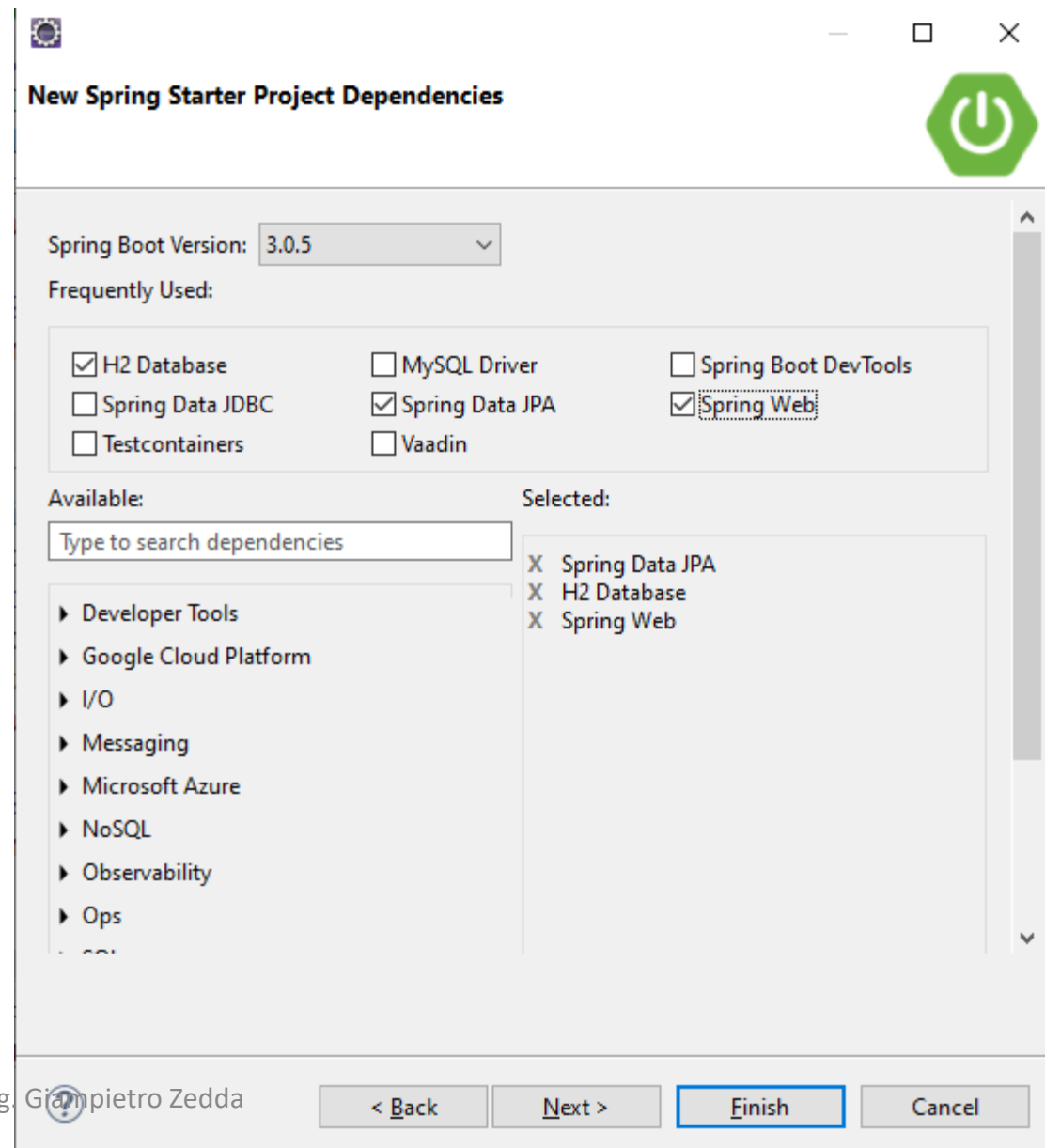
Package:

Working sets

☐ Add project to working sets

Working sets:

24/04/2023



New Spring Starter Project Dependencies

Spring Boot Version:

Frequently Used:


<input checked="" type="checkbox"/> H2 Database	<input type="checkbox"/> MySQL Driver	<input type="checkbox"/> Spring Boot DevTools
<input type="checkbox"/> Spring Data JDBC	<input checked="" type="checkbox"/> Spring Data JPA	<input checked="" type="checkbox"/> Spring Web
<input type="checkbox"/> Testcontainers	<input type="checkbox"/> Vaadin	

Available:

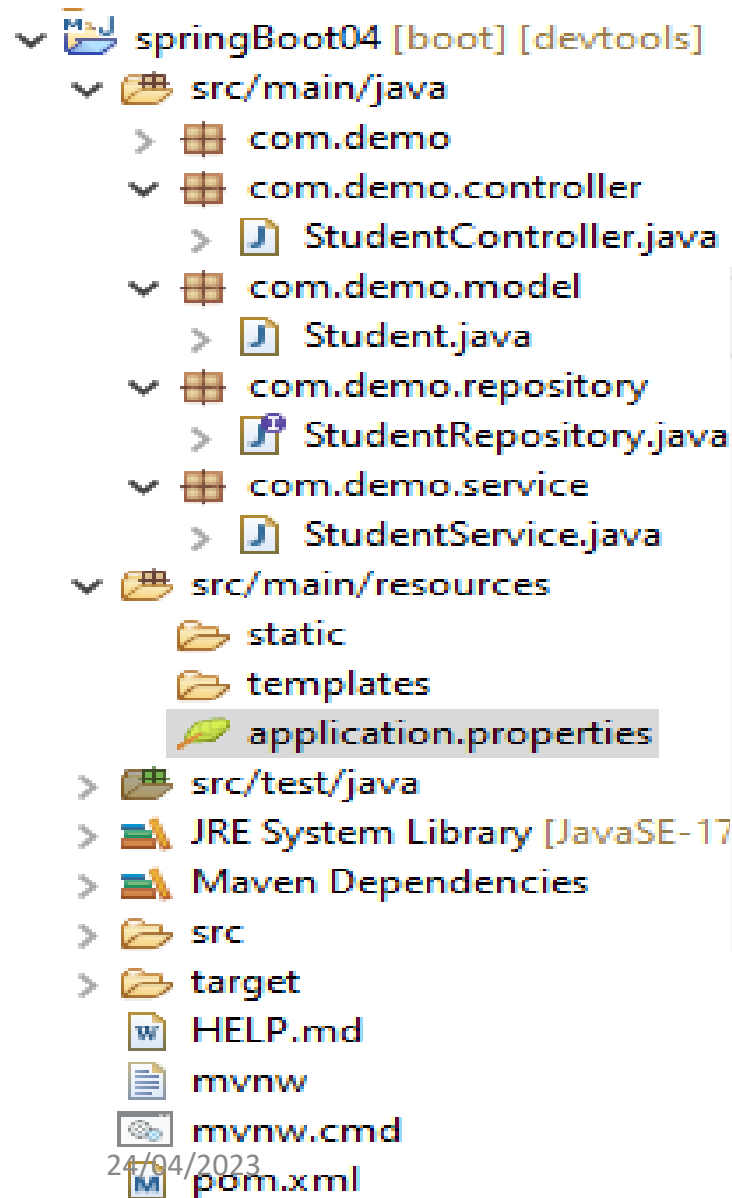
- Developer Tools
- Google Cloud Platform
- I/O
- Messaging
- Microsoft Azure
- NoSQL
- Observability
- Ops

Selected:

- X Spring Data JPA
- X H2 Database
- X Spring Web

ing.  Gianpietro Zedda

Spring Boot H2 database Properties



```
springBoot0... application.... X springBoot0... Student.java StudentCont... Stude
1 spring.datasource.url=jdbc:h2:mem:spformazione
2 spring.datasource.driverClassName=org.h2.Driver
3 spring.datasource.username=sa
4 spring.datasource.password=
5 spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
6 #enabling the H2 console
7 spring.h2.console.enabled=true
8
```

Spring Boot H2 database Model

```
import jakarta.persistence.Id;
import jakarta.persistence.Table;
//Mark class as an Entity
//Defining class name as Table name
@Entity
@Table
public class Student {
    @Id
    @Column
    private int id;

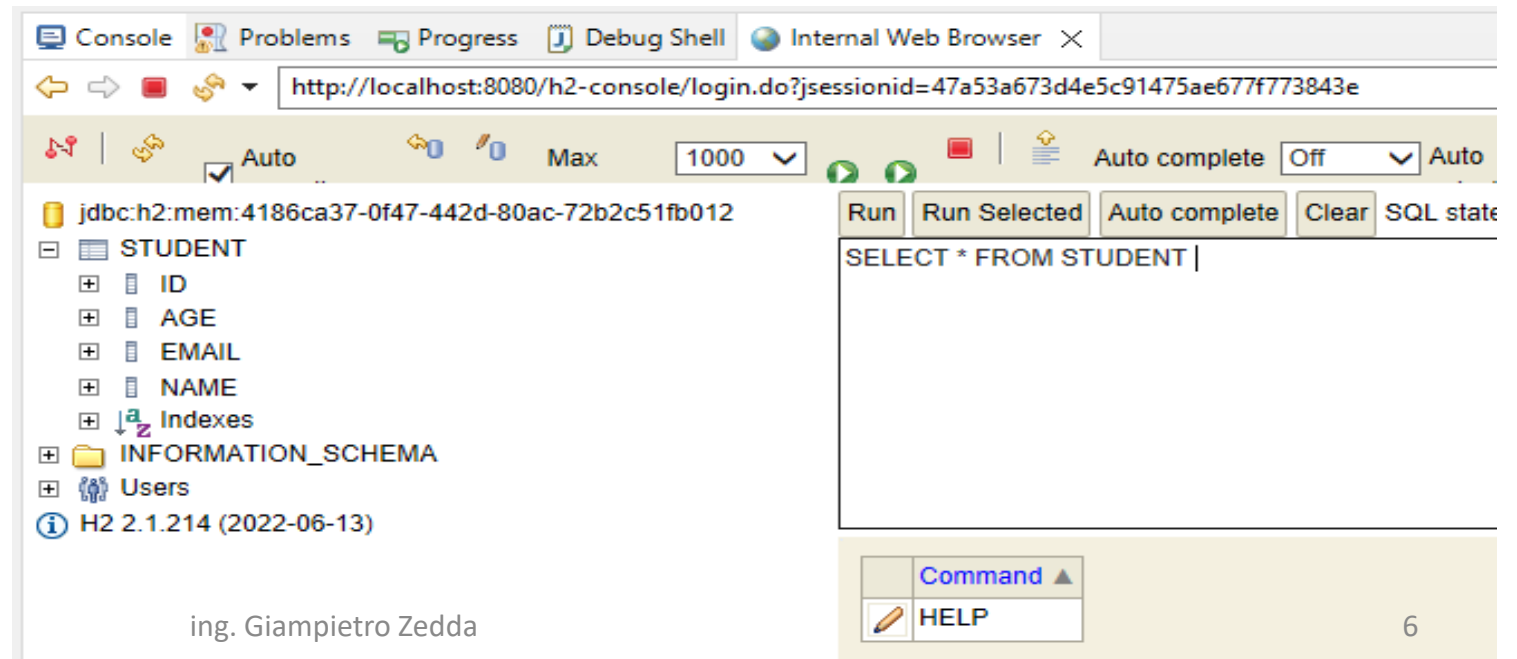
    @Column
    private String name;

    @Column
    private int age;

    @Column
    private String email;

    Public int getId() {
    return id;
}
}
```

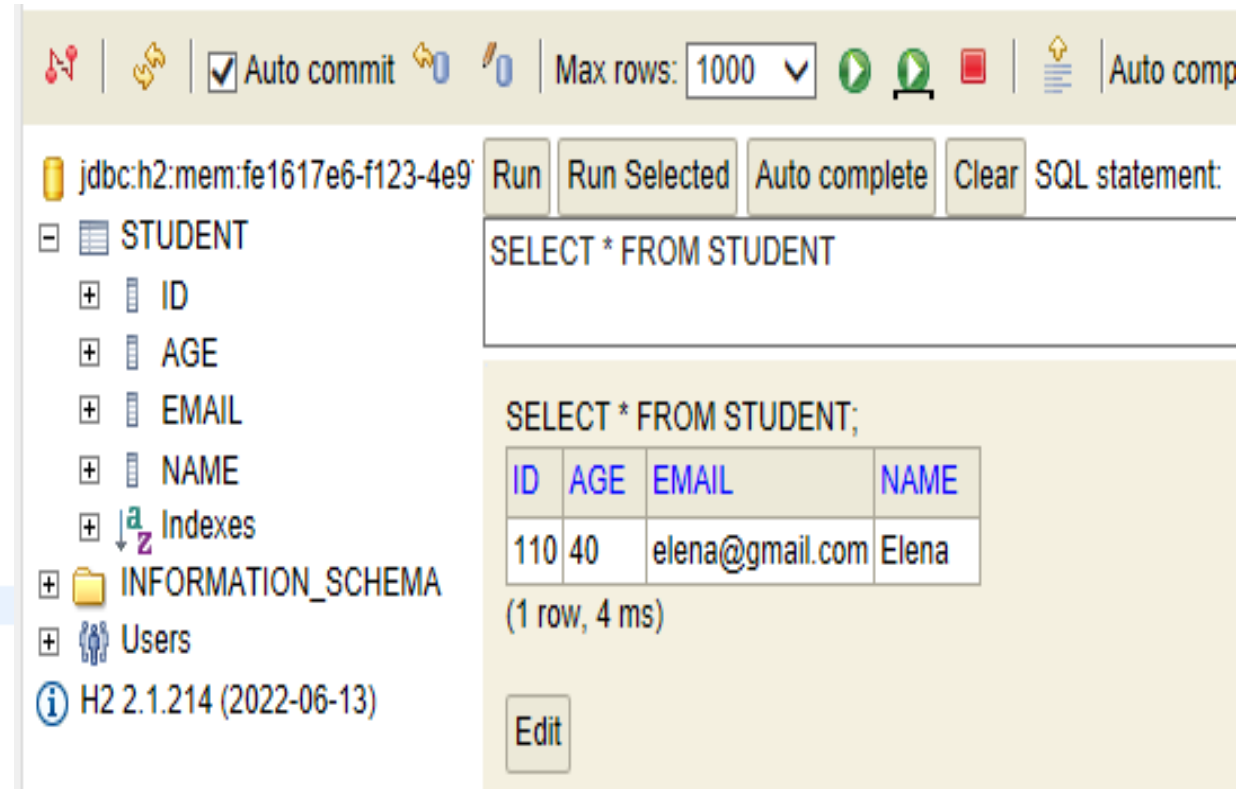
- ❑ H2 attraverso h2-console permette di gestire il database in memory
- ❑ Il database viene generato automaticamente allo startup dell'applicazione via CommandLineRunner



Spring Boot H2 e CommandLineRunner

- ❑ Il database H2 può essere inizializzato allo **startup** dell'applicazione
- ❑ Il codice di inizializzazione, caricamento tabelle etc, può essere codificato nel metodo `run()` implementando l'interface **CommandLineRunner**

```
9 @SpringBootApplication
10 public class SpringBoot04Application implements CommandLineRunner {
11
12     @Autowired
13     private JdbcTemplate jdbcTemplate;
14
15     public static void main(String[] args) {
16         SpringApplication.run(SpringBoot04Application.class, args);
17     }
18
19     @Override
20     public void run(String... args) throws Exception {
21         String sql = "INSERT INTO student (id, age, Name, email) VALUES (?, ?, ?, ?)";
22         int result = jdbcTemplate.update(sql, "110", "40", "Elena", "elena@gmail.com");
23         if (result > 0) {
24             System.out.println("New Row Inserted - 110 Elena");
25         }
26     }
27 }
28 }
```



jdbc:h2:mem:fe1617e6-f123-4e9

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM STUDENT

SELECT * FROM STUDENT;

ID	AGE	EMAIL	NAME
110	40	elena@gmail.com	Elena

(1 row, 4 ms)

Edit

STUDENT

- ID
- AGE
- EMAIL
- NAME
- Indexes
- INFORMATION_SCHEMA
- Users

H2 2.1.214 (2022-06-13)

Spring Boot H2 database Controller

```
@RestController
```

```
public class StudentController {
```

```
    @Autowired
```

```
    StudentService studentService;
```

```
    //creating a get mapping that retrieves all the students detail from the database
```

```
    @GetMapping("/student")
```

```
    private List<Student> getAllStudent() {
```

```
        return studentService.getAllStudent();
```

```
    }
```

```
    //creating a get mapping that retrieves the detail of a specific student
```

```
    @GetMapping("/student/{id}")
```

```
    private Student getStudent(@PathVariable("id") int id) {
```

```
        return studentService.getStudentById(id);
```

```
    }
```

```
    //creating a delete mapping that deletes a specific student
```

```
    @DeleteMapping("/student/{id}")
```

```
    private void deleteStudent(@PathVariable("id") int id) {
```

```
        studentService.delete(id);
```

```
    }
```

```
    //creating post mapping that post the student detail in the database
```

```
    @PostMapping("/student")
```

```
    private int saveStudent(@RequestBody Student student) {
```

```
        studentService.saveOrUpdate(student);
```

```
        return student.getId();
```

```
    }
```

```
24/04/2023
```

```
ing. Giampietro Zedda
```


Spring Boot H2 database Service

```
//defining the business logic
```

```
@Service
```

```
public class StudentService {
```

```
@Autowired
```

```
StudentRepository studentRepository;
```

```
//getting all student records
```

```
public List<Student> getAllStudent() {
```

```
    List<Student> students = new ArrayList<Student>();
```

```
    studentRepository.findAll().forEach(student -> students.add(student));
```

```
    return students;
```

```
}
```

```
//getting a specific record
```

```
public Student getStudentById(int id) {
```

```
    return studentRepository.findById(id).get();
```

```
}
```

```
public void saveOrUpdate(Student student) {
```

```
    studentRepository.save(student);
```

```
}
```

```
//deleting a specific record
```

```
public void delete(int id) {
```

```
    studentRepository.deleteById(id);
```

```
}
```

```
}
```

```
package com.demo.repository;
```

```
import org.springframework.data.repository.CrudRepository;
```

```
import com.demo.model.Student;
```

```
public interface StudentRepository extends
```

```
    CrudRepository<Student, Integer> {}
```

Spring Boot H2 database Postman POST

```
{
  "id": "001",
  "age": "23",
  "name": "Amit",
  "email": "amit@yahoo.co.in"
}
{
  "id": "002",
  "age": "24",
  "name": "Vadik",
  "email": "vadik@yahoo.co.in"
}
{
  "id": "003",
  "age": "21",
  "name": "Prateek",
  "email": "prateek@yahoo.co.in"
}
```

The screenshot displays a Postman interface for a POST request to `http://localhost:8080/student`. The request body is a JSON array of three student objects. Below the Postman interface, the H2 console shows the execution of the query `SELECT * FROM STUDENT`, resulting in a table with three rows of student data.

Postman Request Details:

- Method: POST
- URL: `http://localhost:8080/student`
- Body Type: JSON
- Body Content:

```
1
2 {
3   "id": "001",
4   "age": "23",
5   "name": "Amit",
6   "email": "amit@yahoo.co.in"
7 }
```

H2 Console Output:

Query: `SELECT * FROM STUDENT`

ID	AGE	EMAIL	NAME
1	23	amit@yahoo.co.in	Amit
2	24	vadik@yahoo.co.in	Vadik
3	21	prateek@yahoo.co.in	Prateek

(3 rows, 1 ms)

Spring Boot H2 database Postman GET all

The screenshot shows the Postman interface for a GET request to `http://localhost:8080/student`. The request is successful, returning a 200 OK status with a response time of 186 ms and a size of 350 B. The response body is displayed in JSON format, showing a list of three student records.

Request:

- Method: GET
- URL: `http://localhost:8080/student`
- Body Type: JSON

Response:

```
[{"id":1,"name":"Amit","age":23,"email":"amit@yahoo.co.in"}, {"id":2,"name":"Vadik","age":24,"email":"vadik@yahoo.co.in"}, {"id":3,"name":"Prateek","age":21,"email":"prateek@yahoo.co.in"}]
```

Spring Boot H2 database Postman GET 2

The image shows the Postman interface for a GET request. The URL is `http://localhost:8080/student/2`. The request body is a JSON object: `{ "id": 2, "name": "Vadik", "age": 24, "email": "vadik@yahoo.co.in" }`. The response status is 200 OK, with a time of 32 ms and a size of 224 B. The response body is displayed in a pretty JSON format.

GET `http://localhost:8080/student/2` **Send**

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

```
1  
2 {  
3   "id": 2,  
4   "name": "Vadik",  
5   "age": 24,  
6   "email": "vadik@yahoo.co.in"
```

Body Cookies Headers (5) Test Results Status: 200 OK Time: 32 ms Size: 224 B **Save**

Pretty Raw Preview Visualize JSON

```
1 {  
2   "id": 2,  
3   "name": "Vadik",  
4   "age": 24,  
5   "email": "vadik@yahoo.co.in"  
6 }
```

Spring Boot H2 database Postman DELETE 1

DELETE

http://localhost:8080/student/1

Send

ParamsAuthorizationHeaders (9)BodyPre-request ScriptTestsSettings

● none● form-data● x-www-form-urlencoded● raw● binary● GraphQLJSON

1

2

3

BodyCookiesHeaders (4)Test Results

⌚ Status: 200 OKTime: 21 msSize: 123 BSave

PrettyRawPreviewVisualizeText

1

jdbc:h2:mem:4186ca37-0f47-442

RunRun SelectedAuto completeClearSQL statement:

STUDENT

ID

AGE

EMAIL

NAME

Indexes

INFORMATION_SCHEMA

Users

H2 2.1.214 (2022-06-13)

24/04/2023

SELECT * FROM STUDENT

SELECT * FROM STUDENT;

ID	AGE	EMAIL	NAME
2	24	vadik@yahoo.co.in	Vadik
3	21	prateek@yahoo.co.in	Prateek

(2 rows, 0 ms)

Edit

ing. Giampietro Zedda

Spring Boot H2 database Postman POST 3

POST http://localhost:8080/student

Params Authorization Headers (9) **Body** Pre-request Script Tests Settings

● none ● form-data ● x-www-form-urlencoded ● **raw** ● binary ● GraphQL **JSON** ▼

```
1 {
2   "id": "003",
3   "age": "99",
4   "name": "Giampietro",
5   "email": "gzedda@libero.it"
6 }
7
```

jdbc:h2:mem:4186ca37-0f47-442

STUDENT

- ID
- AGE
- EMAIL
- NAME
- Indexes

INFORMATION_SCHEMA

Users

H2 2.1.214 (2022-06-13)

Run Run Selected Auto complete Clear SQL s

SELECT * FROM STUDENT |

SELECT * FROM STUDENT;

ID	AGE	EMAIL	NAME
2	24	vadik@yahoo.co.in	Vadik
3	99	gzedda@libero.it	Giampietro

(2 rows, 0 ms)

Edit