Spring Boot - How to Access Database using Spring Data JPA

Last Updated : 27 Aug, 2025

Spring Data JPA simplifies database access by providing ready-to-use repositories for CRUD (Create, Read, Update, Delete) operations, reducing boilerplate code. In this article, we'll demonstrate how to connect a database with a Spring Boot application using Spring Data JPA.

Step-by-Step Implementation

Step 1: Create a Spring Boot project

To start, go to the spring initializr and create a new project with the following dependencies:

- **Spring Web:** For building REST APIs.
- **Spring Data JPA**: For interacting with the database using JPA.
- **MySQL Driver**: To connect to MySQL.

Download the generated starter project and import it into the preferred Integrated Development Environment (IDE).

Step 2: Define the Entity Class

Now, we will create a model class named Company and annotate it with @Entity. This annotation indicates that the class is mapped to a table in the database. Add the data members of the class, ensuring their data types match the columns in the database table. Add the annotation @Id to the data member which will behave as the primary key attribute in the table and @GeneratedValue(strategy = GenerationType.AUTO) to auto increment the primary key attribute.

Company.java:

```
package com.example.demo.model;

import jakarta.persistence.*;

@Entity
@Table(name = "companies")
public class Company {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY) // MySQL-friendly
    private Integer id;

    private String name;
    private Integer duration;
    private String profile;
    private Integer stipend;

    @Column(name = "work_from_home")
    private Boolean workFromHome;

    public Company() {}

    public Company(String name, Integer duration, String profile, Integer
stipend, Boolean workFromHome) {
        this.name = name;
        this.duration = duration;
        this.profile = profile;
        this.stipend = stipend;
        this.workFromHome = workFromHome;
    }

    // getters & setters
    public Integer getId() { return id; }
    public void setId(Integer id) { this.id = id; }

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

    public Integer getDuration() { return duration; }
    public void setDuration(Integer duration) { this.duration = duration; }

    public String getProfile() { return profile; }
    public void setProfile(String profile) { this.profile = profile; }

    public Integer getStipend() { return stipend; }
    public void setStipend(Integer stipend) { this.stipend = stipend; }

    public Boolean getWorkFromHome() { return workFromHome; }
    public void setWorkFromHome(Boolean workFromHome) { this.workFromHome =
workFromHome; }
}
```

Key Points:

- The @Entity annotation marks this class as a JPA entity.
- The @Id annotation specifies the primary key.
- The @GeneratedValue(strategy = GenerationType.AUTO) annotation auto-generates the primary key value.

Step 3: Create the Repository Interface

Now, we will create a repository interface CompanyRepository with the annotation @Repository that extends the CrudRepository. This interface will provide CRUD operations for the Company entity.

CompanyRepository.java:

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

import com.example.demo.model.Company;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

@Repository
public interface CompanyRepository extends JpaRepository<Company, Integer> {
}
```

Key Points:

- The @Repository annotation marks this interface as a Spring Data repository.
- CrudRepository<Company, Long> provides built-in methods like save(), findById(), findAll() and deleteById().

Step 4: Configure the Database Connection

Open the application.properties file and add the following configuration to connect to the MySQL database

> *spring.datasource.url=jdbc:mysql://localhost:3306/database_name*
> *spring.datasource.username=root*

```
spring.datasource.password=your_password
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

Key Points:

- Replace database_name with the name of your database.
- Replace your_password with your MySQL password.
- The spring.jpa.hibernate.ddl-auto=update property automatically creates or updates the database schema based on your entity classes.

Step 5: Create REST APIs

Now, we will create REST APIs to perform CRUD operations on the Company entity.

CompanyController.java:

```
package com.example.demo.controller;

import com.example.demo.model.Company;
import com.example.demo.repository.CompanyRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.server.ResponseStatusException;

import java.util.List;

@RestController
@RequestMapping("/companies")
public class CompanyController {

    @Autowired
    private CompanyRepository companyRepository;

    // Home Page
    @GetMapping("/")
    public String welcome() {
        return "<html><body><h1>WELCOME</h1></body></html>";
    }

    // Get All Companies
    @GetMapping
    public List<Company> getAllCompanies() {
        return companyRepository.findAll();
    }

    // Get a Company by ID
    @GetMapping("/{id}")
    public Company getCompanyById(@PathVariable Integer id) {
        return companyRepository.findById(id)
                .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Company not found"));
    }

    // Create a Company
    @PostMapping
    @ResponseStatus(HttpStatus.CREATED)
    public Company createCompany(@RequestBody Company company) {
        return companyRepository.save(company);
    }

    // Update a Company
    @PutMapping("/{id}")
    public Company updateCompany(@PathVariable Integer id, @RequestBody
Company companyDetails) {
        Company company = companyRepository.findById(id)
                .orElseThrow(() -> new
ResponseStatusException(HttpStatus.NOT_FOUND, "Company not found"));

        company.setName(companyDetails.getName());
        company.setDuration(companyDetails.getDuration());
        company.setProfile(companyDetails.getProfile());
        company.setStipend(companyDetails.getStipend());
        company.setWorkFromHome(companyDetails.getWorkFromHome());

        return companyRepository.save(company);
    }

    // Delete a Company
    @DeleteMapping("/{id}")
    @ResponseStatus(HttpStatus.NO_CONTENT)
    public void deleteCompany(@PathVariable Integer id) {
        if (!companyRepository.existsById(id)) {
            throw new ResponseStatusException(HttpStatus.NOT_FOUND, "Company
not found");
        }
        companyRepository.deleteById(id);
    }
}
```

Key Points:

- The @RestController annotation marks this class as a controller for handling REST requests.
- The @RequestMapping("/companies") annotation maps all endpoints in this class to the /companies path.
- The @Autowired annotation injects the CompanyRepository bean.

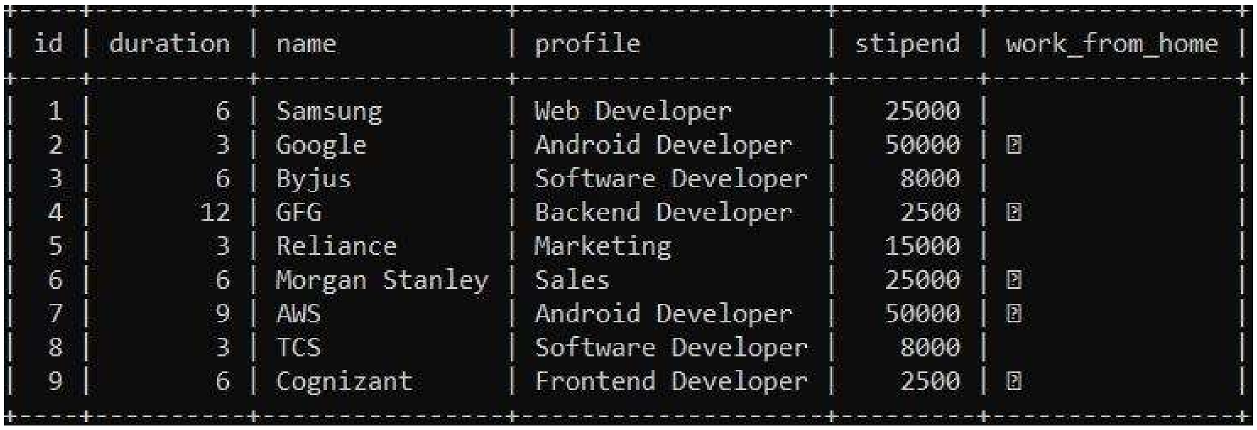
Step 6: Test the Application

Now, we can run the spring boot application. we can test the REST APIs using Postman to verify that the application is functioning as expected

Start MySQL and create DB:

> *CREATE DATABASE database_name;*

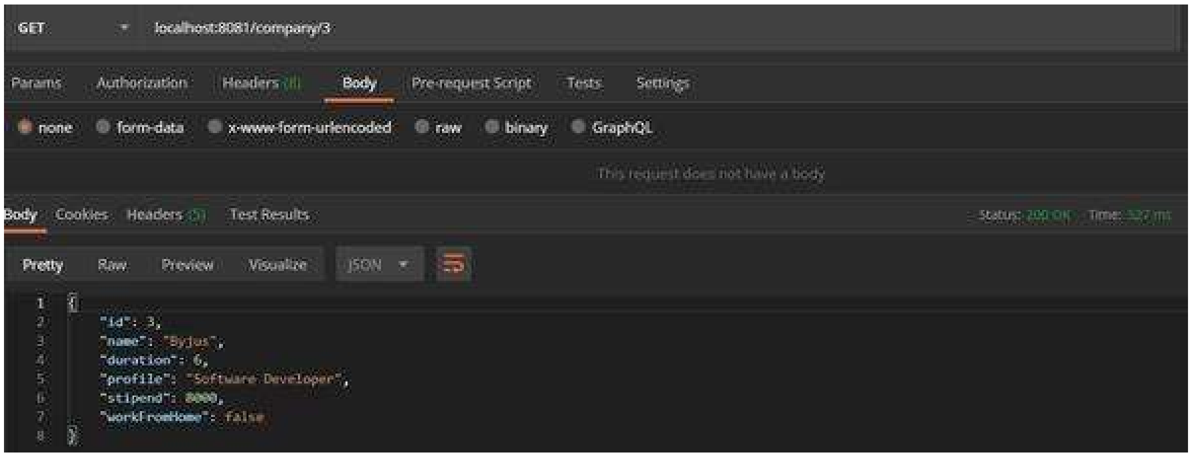
The database:



```
+----+----------+----------------+--------------------+---------+----------------+
| id | duration | name           | profile            | stipend | work_from_home |
+----+----------+----------------+--------------------+---------+----------------+
|  1 |        6 | Samsung        | Web Developer      |   25000 |                |
|  2 |        3 | Google         | Android Developer  |   50000 | □              |
|  3 |        6 | Byjus          | Software Developer |    8000 |                |
|  4 |       12 | GFG            | Backend Developer  |    2500 | □              |
|  5 |        3 | Reliance       | Marketing          |   15000 |                |
|  6 |        6 | Morgan Stanley | Sales              |   25000 | □              |
|  7 |        9 | AWS            | Android Developer  |   50000 | □              |
|  8 |        3 | TCS            | Software Developer |    8000 |                |
|  9 |        6 | Cognizant      | Frontend Developer |    2500 | □              |
+----+----------+----------------+--------------------+---------+----------------+
```

The database

Testing with the POSTMAN collection:



```
GET localhost:8081/company/3
```

This request does not have a body

Status: 200 OK Time: 327 ms

```
{
    "id": 3,
    "name": "Byjus",
    "duration": 6,
    "profile": "Software Developer",
    "stipend": 8000,
    "workFromHome": false
}
```

Testing with the POSTMAN

Comment More info Advertise with us