1.create new java maven spring boot application 2. create RestAPI 3.connect oracle database 4.test postman 5.step by step full process document

**ChatGPT said:**

Thought for 9s

Nice — got you a full, copy-pasteable step-by-step document created and saved: **“Spring Boot + Oracle REST API — Step-by-Step Guide.”** I included everything: project generation, pom.xml hints, Oracle JDBC notes, entities, repository, controller, application.properties, Postman test examples, troubleshooting, and production tips.

You’ll find that guide ready to read and iterate on. Want me to now:

1. Paste a complete pom.xml configured for a specific Spring Boot version (tell me which version),
2. Generate a ready-to-run sample project archive (code files + pom.xml) you can download, or
3. Add a Docker Compose setup for a local Oracle XE for testing?

Pick one and I’ll drop the exact files/code — no waiting.

**You said:**

https://www.springboottutorial.com/spring-boot-with-mysql-and-oracle

**ChatGPT said:**

✅ Perfect — that link is a **solid starting point**.  
But let’s make it **fully Oracle-specific** and modern (using **Spring Boot 3.x + Java 17**).  
Below is your **step-by-step complete process document** — clean, production-ready, and Postman-testable.

**🧩 Step 1: Create a New Spring Boot Project**

Go to 👉 <https://start.spring.io>

**Settings:**

| **Setting** | **Value** |
| --- | --- |
| **Project** | Maven |
| **Language** | Java |
| **Spring Boot** | 3.3.5 (latest) |
| **Packaging** | Jar |
| **Java** | 17 |
| **Group** | com.example |
| **Artifact** | oracle-demo |
| **Name** | oracle-demo |
| **Description** | Spring Boot + Oracle REST API |
| **Dependencies** | Spring Web, Spring Data JPA, Oracle Driver, Lombok |

Then click **Generate** → extract → open in **IntelliJ / Eclipse / VS Code**.

**🏗️ Step 2: Project Structure**

After import, your structure should look like this:

oracle-demo/

├── src/

│ ├── main/

│ │ ├── java/com/example/oracledemo/

│ │ │ ├── OracleDemoApplication.java

│ │ │ ├── controller/

│ │ │ │ └── EmployeeController.java

│ │ │ ├── entity/

│ │ │ │ └── Employee.java

│ │ │ ├── repository/

│ │ │ │ └── EmployeeRepository.java

│ │ │ └── service/

│ │ │ └── EmployeeService.java

│ │ └── resources/

│ │ ├── application.properties

│ │ └── data.sql (optional)

├── pom.xml

**⚙️ Step 3: Configure Oracle Connection**

**src/main/resources/application.properties**

spring.datasource.url=jdbc:oracle:thin:@localhost:1521:ORCL

spring.datasource.username=your\_oracle\_user

spring.datasource.password=your\_oracle\_password

spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver

# Hibernate Config

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.OracleDialect

# Server Port

server.port=8080

✅ **Note:**  
Replace ORCL with your actual Oracle SID or Service Name.  
If using a service name, the URL looks like this:

spring.datasource.url=jdbc:oracle:thin:@//localhost:1521/xe

**💾 Step 4: Create Entity Class**

**Employee.java**

package com.example.oracledemo.entity;

import jakarta.persistence.\*;

import lombok.Data;

@Entity

@Table(name = "EMPLOYEE")

@Data

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(name = "NAME", nullable = false)

private String name;

@Column(name = "DEPARTMENT")

private String department;

@Column(name = "SALARY")

private Double salary;

}

**🧠 Step 5: Create Repository Interface**

**EmployeeRepository.java**

package com.example.oracledemo.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.example.oracledemo.entity.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

**🧩 Step 6: Create Service Layer (Optional but Recommended)**

**EmployeeService.java**

package com.example.oracledemo.service;

import com.example.oracledemo.entity.Employee;

import com.example.oracledemo.repository.EmployeeRepository;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class EmployeeService {

private final EmployeeRepository repo;

public EmployeeService(EmployeeRepository repo) {

this.repo = repo;

}

public List<Employee> getAll() {

return repo.findAll();

}

public Employee save(Employee employee) {

return repo.save(employee);

}

public Employee getById(Long id) {

return repo.findById(id).orElse(null);

}

public void delete(Long id) {

repo.deleteById(id);

}

}

**🌐 Step 7: Create REST Controller**

**EmployeeController.java**

package com.example.oracledemo.controller;

import com.example.oracledemo.entity.Employee;

import com.example.oracledemo.service.EmployeeService;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

private final EmployeeService service;

public EmployeeController(EmployeeService service) {

this.service = service;

}

@GetMapping

public List<Employee> getAll() {

return service.getAll();

}

@PostMapping

public Employee create(@RequestBody Employee emp) {

return service.save(emp);

}

@GetMapping("/{id}")

public Employee getById(@PathVariable Long id) {

return service.getById(id);

}

@DeleteMapping("/{id}")

public String delete(@PathVariable Long id) {

service.delete(id);

return "Deleted Employee with ID " + id;

}

}

**🧪 Step 8: Test with Postman**

Run the application:

mvn spring-boot:run

**Base URL:**

http://localhost:8080/api/employees

**Test Endpoints:**

| **Method** | **URL** | **Description** | **Body (JSON)** |
| --- | --- | --- | --- |
| **POST** | /api/employees | Create new employee | { "name": "Saiful", "department": "IT", "salary": 55000 } |
| **GET** | /api/employees | Get all employees | — |
| **GET** | /api/employees/1 | Get by ID | — |
| **DELETE** | /api/employees/1 | Delete employee | — |

If everything’s wired correctly — you’ll see Oracle table EMPLOYEE auto-created and data inserted ✅.

**🧰 Step 9: Verify in Oracle**

Run:

SELECT \* FROM EMPLOYEE;

You should see the same data created from Postman.

**🧠 Bonus Tips**

* To auto-create schema from SQL file:  
  Create data.sql in resources/ folder.
* To log SQL parameters, add:
* logging.level.org.hibernate.type.descriptor.sql=TRACE

Would you like me to create a **Word/PDF version** of this guide (formatted and ready to share internally)? I can export it neatly for your documentation or team wiki.

Top of Form

Bottom of Form