1) Map directly to student table in MySQL

(ii) @Id +@ Generated value handles primary key

3. Repository Interface

```
import org.spring framework.data.jpa.repository.
        JPA Repository;
public interface Student Repository
        extends JPa Repository <Student, Long> {

}
```

(i) No method implementation needed

(ii) Spring Data JPA automatically provides CPUD method.

4. CPUD Operation using Repository

CREATE - Insert a student

```
Student Repository.save (Student);
```

(i) if id is null, JPA perform INSERT

(ii) Used for creating new student records.

READ - Fetch Student data

Read all students

List <Student> students = StudentRepository.findAll();

Read by ID

Student student = StudentRepository.findById(id).

orElse(null);

findAll() → SELECT *

findById() → SELECT where id = ?

Update - Modify Student Record

Student, Student = StudentRepository.findById(id).

orElse(int);

Student.setMarks(90);

StudentRepository.save(Student);

(i) if id exit → JPA perform update

(ii) Same save() method handles both create an update.

(iii) DELETE - Remove Student

StudentRepository.deleteById(id);

(i) Delete record using primary key

```
rs.close();
Stmt.close();
Con.close();
}
catch (Exception e){
        e.printStackTrace();
    }
  }
}
```

Output:

```
1  Afrin  85
2  Mili   80
```

26 Design a simple CRUD application using spring Boot and MySQL to manage student record, Describe how each operation (Create, Read, Update, Delete) would be implemented using a repository interface.

A CRUD application perform four basic operations.

Create – Insert new data

Read – Retrieve existing data

Update – Modify existing data

Delete – Remove data

In Spring Boot, CRUD operations are easily implemented using Spring Data JPA and a repository interface, withe reduce boilen plate JDBe code.

Overall Structure

Client (Browser / postman)

↓

Controller (REST API)

↓

Service Layer (optional but recommanded)

↓

Repository Interface (Jpa repository)

↓

MySQL Database

2. Student Entity (Model

```
import jakarta.persistence.*;
@ Entity
@ Table (name = "student").
public class Student {
 @ Id
 @ Generated value ( strategy = Generationtype.IDENTITY)
     private long id;
     private String Name;
     private int marks;
     // getters and setters
}
```

(ii) Internally executes DELETE query.

5. Controller example

```
@ Restcontroller
@ Request mapping ("/students")
public class student Controller {
    @ Autowired
    private student Repository repository;
    @ post Mapping
    Public student create(@ requestBody student s){
    return repository.save(s);
    }
@ getMapping
    public List< student > getAll () {
    return Repository findAll();
    }
@ putMapping ("/{id}")
public student update @ path variable Lang id,
            @ Request Body student s) {
```

```
    s.setId(id);
    return repository.save(s);
}
@DeleteMapping("/{id}")
public void delete @PathVariable Long id){
    repository.deleteById(id);
}
```

6. Application properties

```
spring.datasource.url = jdbc:mysql://localhost:3306/
spring.datastructure.username = root
spring.data source.password =
spring.data jpa.hibernate.ddl=auto=update
spring.jpa.show-sql=true.
```

Operation	Repository Method
Create	Save()
Read (All)	findAll()
Read (ById)	findById()
Update	Save()
Delete	deleteById()

27. How does Spring Boot simplify the development of RESTful service? Describe how to implement a REST Controller using @RestController, @GetMapping and @PostMapping, including JSON data handling.

Spring Boot reduces complexity by providing:

Auto-Configuration

- Automatically configures web server
- NO XML Configuration needed

Embedded server

- Built-in tomcat → no external server setup

Starter Dependencies

- Spring-boot-starter-web includes:
- Spring MVC
- REST support
- Jackson (JSON)

Annotation-Based Development

- Simple annotations like @RestController, @GetMapping, @PostMapping

REST Controller Basics

Annotation	Purpose
@RestController	Marks class as REST controller
@GetMapping	Handles HTTP GET Requests
@PostMapping	Handles HTTP POST requests
@RequestBody	Converts JSON → Java object
@ResponseBody	Converts Java object → JSON

Example : Simple REST API with JSON

1. Model class (Student.java)

```
public class Student {
    private int id;
    private String name;
    private double marks;
    public Student() { }
```

```
public Student (int id, string name, double marks){
        this.id = id;
        this.name = name;
        this.marks = marks;
}
  public int getId() {return id;}
  public void setId (int id) {this.id = id;}
  public String getName() { return name;}
  public void setName (String name) {this.name=name;}
  Public double getMarks () { return marks;}
  public void setMarks (double marks){ this.mark
                                    = marks;}
}
```

2. REST Controller (StudentController.java)

```
import org.springframework.web.bind.annotation.*;
import java.util.*;
@RestController
@RequestMapping ("/students")
```

```
public class StudentController {
    private List<Student> student = new ArrayList<>();

@ GetMapping
  public List<Student> getAllStudents() {
    return students;
  }
@ PostMapping
  public Student addstudent(@ RequestBody Student
                                    Student) {
      Students.add(Student);
      return student;
  }
}
```

JSON handling work.

POST Request (JSON → Java)

```
{"id": 1,
 "name": "Afrin";
  "mark": 88.5 }
```

- @ RequestBody uses Jackson
- Converts JSON into student Object automatically.

GET Response (Java → JSON)

```
{
  "id": 1,
  "Name": "Afrin",
  "mark": 88.5
}
```

- Spring Boot automatically converts objects to JSON
- No manual Serialization needed.

32. Demonstrate the project you developed with the important codes and Graphical User Interface.

The project demonstrates how a Spring Boot Java application communicates with a relational database using JDBC and displays the fetched data in a graphical web interface.

Backend: Spring Boot + JDBC

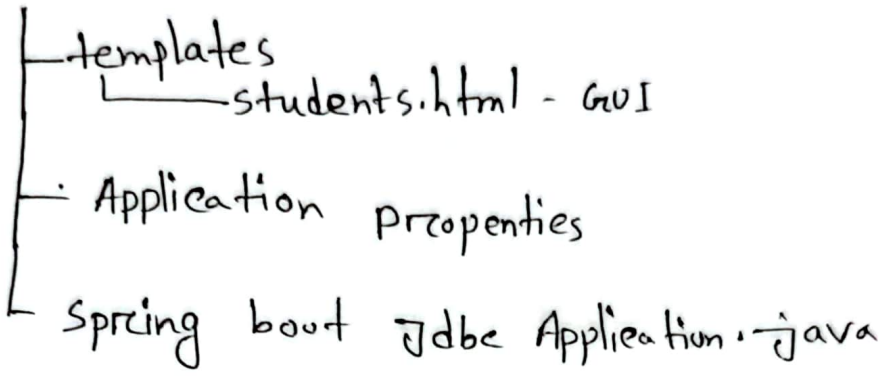
fronted (GUI) : HTML + thymeleaf

Database : MySQL

Operation : Execute SELECT query and show result in a table.

Update project Structure

```
Spring boot - Jdbc - project
    ├── controllers
    │     └── student Controllers.java
    ├── Model
    │     └── student.java
    ├── Repository. student Repository.java
```



Student Repository.java

```
@ Repository
public class Student Repository {
    public List<student> get All Students() {
     List<student> list = new. ArrayList<>();
    try {
    Connection con = Drivere Manager.get Connection(
        " jdbc :mysql://localhost:3306/student db";
        " root",
        " password"
       );
    Statement stmt = con. create statement();
    result set rs = stmt. execute Qurrery (
    " SELECT id, name, cgpa From students"
    ) {
```

```
while (rs.next()){
        list.add(new Student(
                rs.getInt("id");
                rs.getString("name");
                rs.getDouble("cgpa");
            );
    } catch (SQLException e){
            e.printStackTrace();
    } return list;
  }
}
```

2. Controller connection Backend to GUI Student

Controller.java

```
@ Controller
  public class student Controller{
  private final Student Repository reposity;
  Public Student controller(Student Repository repository)
      { this.repository = repository;
  }
```

```
@GetMapping ("/students")
  public String viewStudents (Model model){
      model.addAttribute ("Students").
        repository.getAll students ());
      return "student";
}
}
```

3. Graphical user Interface

student.html

```
<! DOCTYPE html>
<html xmlns:th = "http://www.thymeleaf.org">
<head>
    <tittle> student List </tittle>
    <style>
    body { font-family : Arial;
    background-colour : #f4f6f8;
}
table { bonder-collapse : collapse;
    width : 607;
    mangin : 50px auto;
    background : white;
}
```

```
th, td {
    padding : 12px;
    border : 1px solid #ccc;
    text-align: center;
}
th {
    background-colour: #2c3e50;
    colour: white;
}
h2 { text-align : center;
    margin-top : 40px;
}
</style>
</head>
<body>
<h2> Student Information (Spring Boot JDBC </h2>
<table>
<tr>
<th> &ID </th>
</tr>
<tr th:each = "s: ${students}">
    <td th:text = "${s.id}" ></td>
    <td th:text = "${s.name}" ></td>
```

```
<td th:text = "${s.cGrpA">
</tr>
</table>
</body>
</html>
```

4. GUI output

http://localhost:8080/students

(i) clean web page
(ii) student data displayed in table format
(iii) Data fetch directly form database.

Example :

ID	Name	CGPA
Afi 01	Afrin	3.82
02	Mili	3.80

5. How the whole system works

(i) user opens /students in browser
(ii) controller receives request
(iii) Repository execute JDBC SELECT query
(iv) Data fetched using ResultSet
(v) Data set to thymeleat page
(vi) GUI displays record in table.