24. How does JPA manage the mapping between Java objects and relational tables? Explain it with an Example using @Entity, @Id and @GeneratedValue annotation. Discuss the advantage of using JPA over raw JDBC

Answer:

JPA manage the mapping between Java objects and relational database tables using ORM. It follows developers to work with database data as Java object instead of writing SQL query manually

i) A Java class represent a database table
ii) class fields represent table comumn
iii) Annotation define how the mapping is

done

```
@Entity
public class student{
        @Id
        @GeneratedValue
        private int id;
        private string name;
        private string email;
}
```

25. Desenibe the difference between the EntityManager's persist(), merge(), and remove operations, when would you use each method in a typical database transaction scenario?

Answer:

persist:

i) Used to save a new entity to the database

ii) makes a new object persistent

iii) The entity becomes managed by the Entity manager.

iv) an Insert operation is performed.

merge():

i) Used to update an exsiting entity.

ii) copies data from a detached entity into a managed entity.

iii) Return a managed instance

iv) Perform an UPDATE

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

Answer:

A CRUD application perform four basic operations.

Create - Insent 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, with reduce boilenplate JDBC 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
    @GeneratedValue(strategy = GenerationType.IDENTITY
    private Long id;

    private String Name;

    private int marks;

    // gettens and settens

}
```

i) Map directly to student table in MySQL

ii) @Id + @GeneratedValue handles primary key

3. Repository Interface

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

public interface studentRepository
        extends JpaRepository<student, Long>{

}
```

i) No method implementation needed

ii) Spring Data JPA automatically provides CRUD method

4. CRUD operation Using Repository

CREATE - Insert a student

```
studentRepository.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 = student Repository. 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(null)
student. setMarks (90);
studentRepository. save (student);
```

i) if id exist → JPA perform Update

ii) Same save() method handles both create and update

DELETE - Remove student

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

i) Delete record using primary key

ii) Internally executes DELETE query

5. Controller example

```
@RestController
@RequestMapping("/students")
public class StudentController{

    @Autowired
    private StudentRepository repository;

    @postMapping
    public student create(@requestBody Student s){

     return repository.save(s);
    }

   @getMapping
   public List<Student> getAll(){
     return repository.findAll();
   }
@putMapping("/{id}")
public student update @PathVariable Long id,
                      @RequestBody 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/colleg

spring.datastructure.username = root

spring.datasource.password =

spring.jpa.hibernate.ddl-auto = update

spring.jpa.show-sql = true

Operation	Repository Method
Create	save()
Read(All)	findAll()
Read (By ID)	findById()
Update	save()
Delete	deleteById()

remove()

i) Used to delete an entity from the database.

ii) the entity must be managed

iii) performs a DELETE operation

28. Compare @RestController and @Controller in Spring Boot. In a RESTful API for a library system, describe how you would structure the endpoint for books using REST principle.

Answer:

@Controller is used in Spring MVC to return views, while @RestController is used to build REST APIs and return data directly. In a Library REST API, book resource are exposed using endpoint like GET /book,

29. How does Maven manage dependencies and build lifecycle in a spring Boot project. Explain the structure of a typical pom.xml and how the Spring Boot starter dependency simplify development.

Answer:

Maven manages dependencies and the buid lifecycle in a Spring Boot project through the pom.xml file. The pom.xml defines project information, dependencies and build plugin. Spring Boot starter dependencies group commonly used libraries into a single dependency, reducing configuration and simplify development.

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

Answer:

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

```
springboot-jdbc-project
├── controller
│   └── StudentController.java
├── model
│   └── Student.java
├── Repository
│   └── StudentRepository.java
├── templates
│   └── students.html  <- GUI
├── Application.properties
└── SpringbootJdbcApplication.java
```

studentRepository.java

```
@Repository
public class StudentRepository {

    public List<Student> getAllStudents(){

    List<student> list = new ArrayList<>();

  try{
  Connection con = DriverManager.getConnection(
      "jdbc:mysql://localhost:330/studentdb",
      "root",
      "password"
     );

  Statement stmt= con.createStatement();
  ResultSet rs = stmt.executeQuery(
      "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. Controllen connection Backend to GUI

studentControllen.java

```
@ Controllen
public class studentControllen{
    private final studentRepositony neposity;
    publie StudentControllen(StudentRepository nepositon
        this.nepositony = nepositony;
    }
@GetMapping("/students")
public string viewStudents(Model model){
    model.addAttnibute("students",
            nepositony.getAllstudents());
        return "student";
    }
}
```

3. Graphical User Interface

student.html

```
<!DOCTYPE html>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <title> Student List </title>
    <style>
      body {
        font-family: Arial;
        background-colour: #f4f6f8;
      }
     table{
        bonder-collapse: collapse;
        width: 60%;
        margin: 50px auto;
        background: white;
     }
    th, td{
        padding: 12px;
        bonder: 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>
      <th> Name </th>
      <th> CGPA </th>

   </tr>
   <tr th:each = "s : ${students}">
         <td th:text = "${s.id}" ></td>
         <td th:text = "${s.name}" ></td>
         <td th.text = "${s.CGPA" ></td>

    </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 from database.

Example

ID	Name	CGPA
1	Mimi	3.75
2	Hritu	4.00

5. How the whole system works

1. User opens /students in browser
2. Controller receives request
3. Repository executes JDBC SELECT query
4. Data fetched using ResultSet
5. Data sent to Thymeleaf page
6. GUI displays record in table.

31. You are building a multi-module spring Boot application with separate modules for API, service, and database. Explain how you would structure the project using maven and Gradle and inter module dependencies effectively

Answer:

In a multi-module Spring Boot application, the project is divided into separate modules such as API, service and database to achieve better separation of concern, reusability and maintainability

```
parent-project
├── pom.xml
├── Api-module
│   └── pom.xml
├── service-module
│   └── pom.xml
└── database-module
    └── pom.xml
```

33. socket programming example:

Answer:

Server Program:

```
import java.net.*;
import java.io.*;
public class Server{
    public static void main(String[] args) throws
    Exception{
    ServerSocket serverSocket = new ServerSocket
    Socket socket = serverSocket.accept()
    DataInputStream in = new DataInputStream
    (socket.getInputStream());
    String msg=in.readUTF();
    System.out.println("Client say:"+msg);
    serverSocket.close();
    }
}
```

Client program

```
import java.net.*;
import java.io.*;

public class Client{
    public static void main(String[] args) throws
    Exception{

    Socket socket = new Socket("localhost", 5000);

  DataOutputStream out = new DataOutputStream
    (socket.getOutputStream());

    out.writeUTF("Hellow server");

    socket.close()
    }
}
```

34. RMI Example chat system with FX

Answer:

1. Remote Interface

```
import java.rmi.*;
public interface chatservice extends Remote{
    void sendMessage(String msg) throws
RemoteException;
{
```

2. JavaFX client

```
chatService chat =
(chatservice) Naming.lookup("rmi://localhost
                            /chat");
chat.sendmessage("Hello from FX client
```

3. RMI servers

```
Import java.rmi.*;
import java.rmi.server.*;

public class chatServer extends UnicastRemoteObject
implements chatService{
        protected chatServer() throws RemoteExcept
        }}

public void sendMessage(string msg){
        System.out.println("Client" + msg);

}
```