UNIT IV –Second Part

1. **Accessing inmemory DB with JDBCTemplate**
2. **Accessing inmemory DB with Spring JPA**
3. **Accessing MySQL with Spring JPA**
4. **Query methods in Spring data JPA**

1. JDBCTemplate with In-memory database

Steps to be followed:

1. Add the dependency **h2** in pom.xml
2. Create a class to be saved in the database (here Student)
3. Create a service class to access the database

Creating a Class

```
public class Author {
private String firstName;
private String lastName;
public String getFirstName() {
return firstName;
}
public void setFirstName(String firstName) {
this.firstName = firstName;
}
public String getLastName() {
return lastName;
}
public void setLastName(String lastName) {
this.lastName = lastName;
}
public Author(String firstName, String lastName) {
super();
this.firstName = firstName;
this.lastName = lastName;
}
}
```

variable

setters & getters

Constructor to store values.

Service class:To store and retrieve the records in the database, we can create a service class which autowires the `JdbcTemplate` object into the `jdbcTemplate` variable

```
import java.util.ArrayList;
import java.util.*;
import org.slf4j.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Service;
import javax.annotation.*;
@Service
public class AuthorService {
private static final Logger log =LoggerFactory.getLogger(AuthorService.class); → To print in log.
@Autowired
JdbcTemplate jdbcTemplate; ——→ field injection.
@PostConstruct
public void postConstruct() {
        Author author1=new Author("phani","it");
        Author author2=new Author("ashok","it");
        List <Author> authors = new ArrayList<>(); → List
        authors.add(author1);
        authors.add(author2);
log.info("Creating tables");
        jdbcTemplate.execute("DROP TABLE author IF EXISTS");
        jdbcTemplate.execute("CREATE TABLE author(" + " first_name varchar(255),
last_name varchar(255))");
authors.forEach(i->jdbcTemplate.update("INSERT INTO author(first_name, last_name)
VALUES (?,?)",i.getFirstName(),i.getLastName()));
log.info("Records Saved");
//retrieve saved records.
log.info("Retrieving records");
authors = jdbcTemplate.query("select * from author", (rs, rowNum)-> new
Author(rs.getString("first_name"),rs.getString("last_name")));
authors.forEach(i -> log.info(i.getFirstName() + " " +i.getLastName()));
}
}
```

} creating objects to class
} adding objects to list
} Table creation
} Insertion
} Retrieving
} printing.

2. Spring JPA with In-memory database

- JPA is Java Persistence API
- It is a specification related to saving or persisting Java objects which are required by businesses or applications to be saved
- JPA is just a guideline which all **Object Relational Mapping (ORM)** models should follow
- The dependency to be added is spring-data-jpa

Steps to be followed:

1. Add the dependency **spring-data-jpa** in pom.xml
2. Create a class to be saved in the database (here Student)
3. create an interface that extends CrudRepository to perform CRUD operations on the JPA entity
4. Create a service class to access the database

Class to be saved in database

```
import java.io.*;
import javax.persistence.*;
@Entity
public class Student implements Serializable {
@Id
private long id;
private String name;
public long getId() {
return id;
}
public void setId(long id) {
this.id = id;
}
public String getName() {
return firstName;
}
public void setName(String name) {
this.name =name;
}
public Student(String name) {
super();
this.name=name;
}
public Student(String name) {
super();
this.name=name;
}
@Override
public String toString() {
return "Student [id=" + id + ", Name=" + name + "]";
}}
```

Create an interface

The CrudRepository interface takes the name of the entity and its primary key.
Here, it is **Student** and **ID**

```
import org.springframework.data.repository.CrudRepository;
public interface StudentRepository extends CrudRepository<Student, Long>{
}
```

Service class to access the database

```
import javax.annotation.*;
import org.slf4j.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
@Service
public class StudentService {
private static final Logger log = LoggerFactory.getLogger(StudentService.class);
@Autowired
StudentRepository sr;
@PostConstruct
public void postConstruct() {
Student ob= new Student();
        ob.setId(1L);
        ob.setName("Kumar");
sr.save(ob);
//retreiving
log.info("Student:" + sr.findAll());
}
}
```

3. Spring JPA with MySQL

Steps to be followed:

1. Add the dependency **mysql-connector-java** in pom.xml
2. Create a table Student in MySQL
3. create an interface that extends CrudRepository to perform CRUD operations on the JPA entity
4. Create a service class to access the database

Create Student Table in the database

Id	Name

Create an interface

The CrudRepository interface takes the name of the entity and its primary key.
Here, it is **Student** and **ID**

```
import org.springframework.data.repository.CrudRepository;
public interface StudentRepository extends CrudRepository<Student, Long>
{
}
```

interface

Service class to access the database

```
@Service
public class StudentService {
private static final Logger log = LoggerFactory.getLogger(StudentService.class);
@Autowired
StudentRepository sr;
@PostConstruct
public void postConstruct() {
Student ob= new Student();
ob.setId(1L);
ob.setName("Kumar");
sr.save(ob);
//retreiving
log.info("Student:" + sr.findAll());
}
}
```

log.

field injection

creating object.

Save to db.

Retrieve.

findAll() method of the **CrudRepository** interface to retrieve all records from the database for a given entity.
In MySQL, run the command to check the data is inserted:
select * from Student

4. Query methods in Spring data JPA

Consider, a Student Table

```
public interface StudentRepository extends CrudRepository<Student, Long>
{
List<Student> findByName(String name);
}
```

S. No.	Method Name	Purpose	Example
1	findAll()	Used to retrieve all records from the database	@Autowired StudentRepository sr; sr.findAll()
2	findByName()	Retrieve all records based on the name	List<Student> s = sr.findByName("Kumar");
3	findByFirstName()	Retrieve all records based on the firstname	List<Student> s =sr.findByFirstName("Kumar");
4	findByFirstNameAndLastName()	Retrieve all records based on the firstname and lastname	List<Student> s = r.findByFirstNameAndLastName(String firstName, String lastName);
5	findByFirstNameOrLastName()	Retrieve all records based on the firstname or lastname	List<Student> s = sr.findByFirstNameOrLastName(String firstName, String lastName);
6	findByLastNameOrderByFirstNameAsc()	Ordering the retrieved results based on firstname	List<Student> s =sr.findByLastNameOrderByFirstNameAsc(String lastName
7	findFirst10ByLastname()	Getting first 10 results based on lastname	List<Student> s = sr.findFirst10ByLastname (String lastName);
8	@Query()	To write our own queries	@Query(value = "select * from Student where first_name=?", nativeQuery = true) Student fetchByFirstName(String firstName);