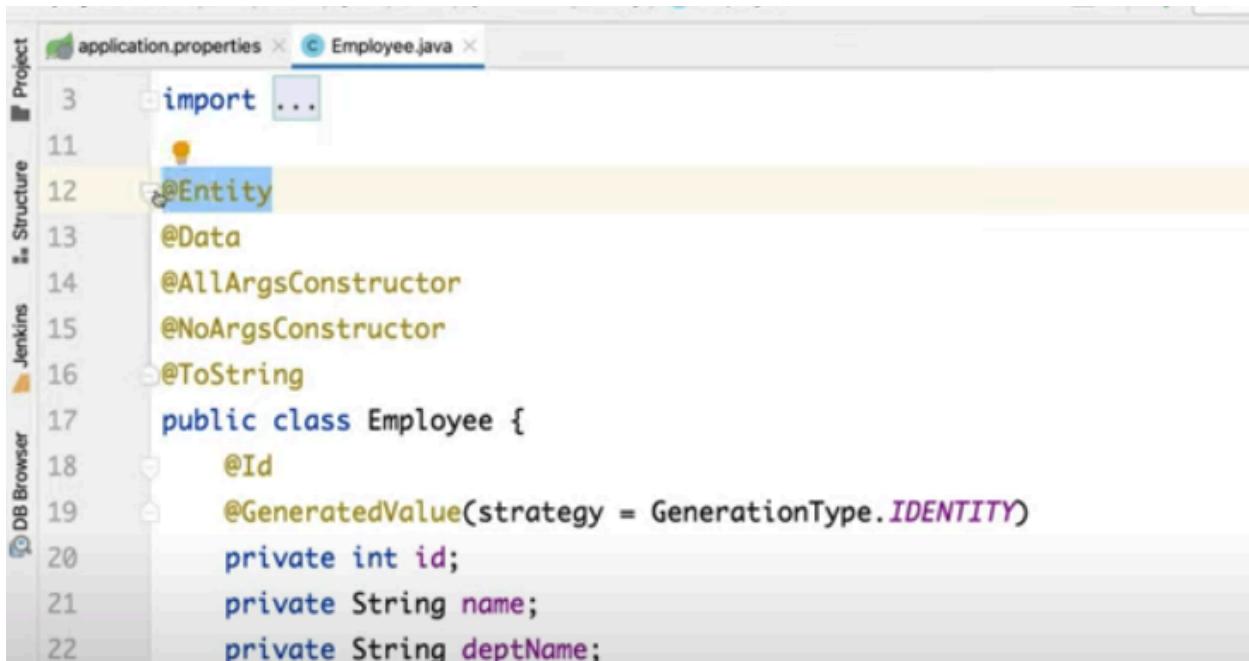


61. How does your application interact with the database and which frameworks are you using?

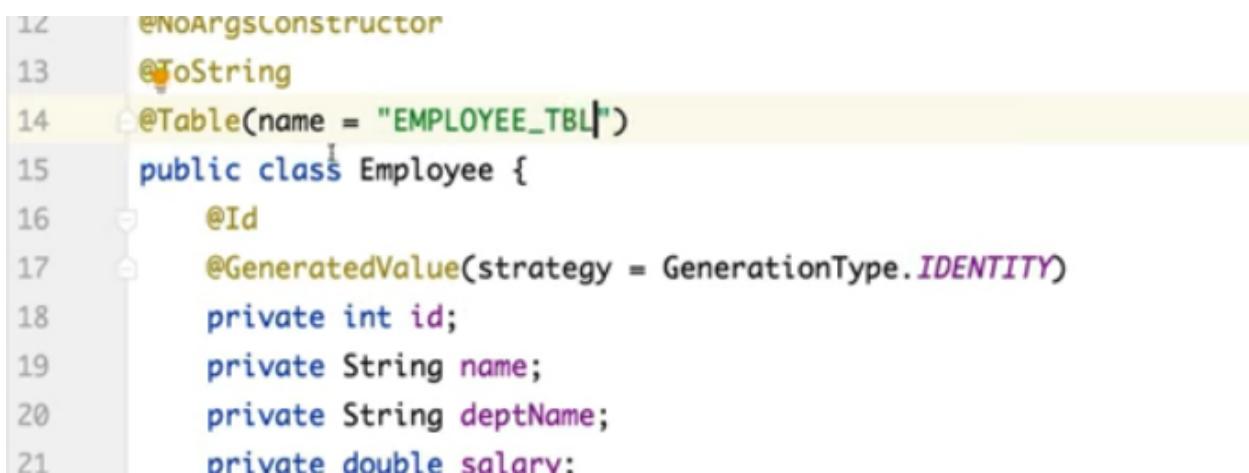
Create entity class. If you want to define any POJO as an entity which we want to store in DB then use this annotation. It will create table with the class name.



The screenshot shows a Java code editor with the file 'Employee.java' open. The code defines a class 'Employee' with annotations: @Entity, @Data, @NoArgsConstructor, @AllArgsConstructor, and @ToString. The class has three private fields: id, name, and deptName. The id field is annotated with @Id and @GeneratedValue(strategy = GenerationType.IDENTITY). The code editor interface includes tabs for 'application.properties' and 'Employee.java', and various toolbars and panels on the left.

```
import ...
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private String deptName;
```

For custom table name use @Table annotation

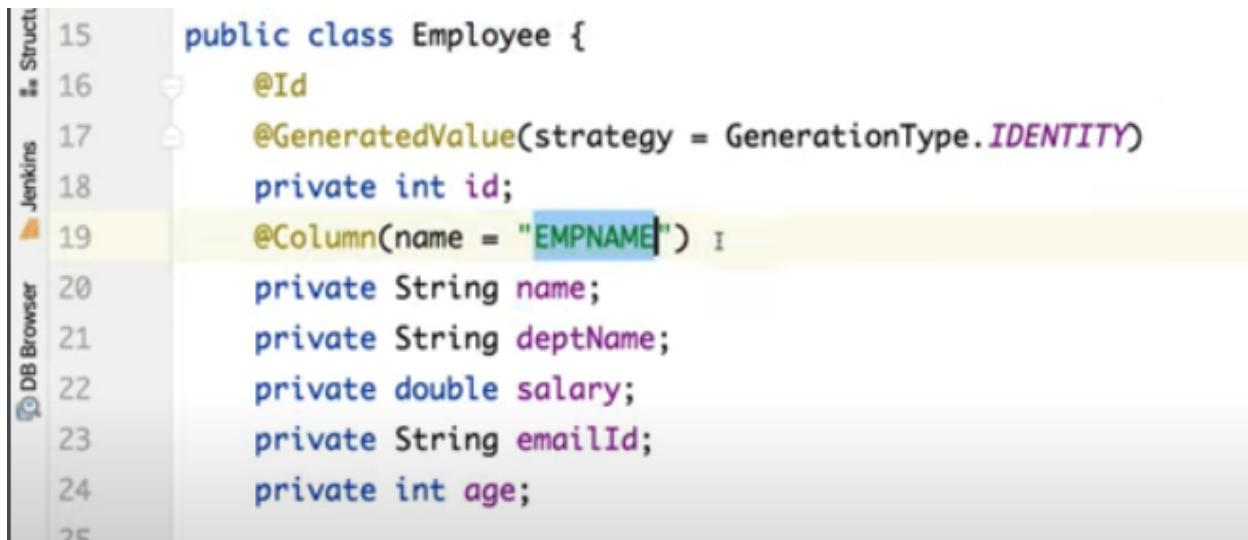


The screenshot shows the same Java code editor with the 'Employee.java' file. The @Table annotation is added to the class definition, specifying a table name of "EMPLOYEE\_TBL". The code editor interface is similar to the previous screenshot.

```
@NoArgsConstructor
@ToString
@Table(name = "EMPLOYEE_TBL")
public class Employee {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;
    private String name;
    private String deptName;
    private double salary;
```

For primary we have to use @Id and @GeneratedValue annotations Identity for auto increment.

For customized column name use @Column annotation.



```
15 public class Employee {  
16     @Id  
17     @GeneratedValue(strategy = GenerationType.IDENTITY)  
18     private int id;  
19     @Column(name = "EMPNAME")  
20     private String name;  
21     private String deptName;  
22     private double salary;  
23     private String emailId;  
24     private int age;  
25 }
```



```
4 #DATASOURCE PROPERTIES  
5 spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
6 spring.datasource.url = jdbc:mysql://localhost:3306/javatechie  
7 spring.datasource.username = root  
8 spring.datasource.password = Password  
9  
10 #JPA SPECIFIC PROPERTIES  
11 spring.jpa.show-sql = true  
12 spring.jpa.hibernate.ddl-auto = update  
13 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect  
14 #spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalN
```



```
12 spring.jpa.hibernate.ddl-auto = create
```

If we use ddl-auto as create, it will always create table.

If we use ddl-auto as an update, it will create a table if it does not exist. If it exists it only updates the table.

Best practice is always update.

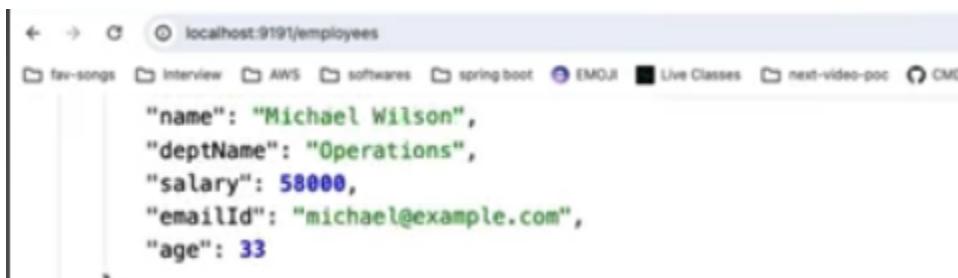
Use create only in integration / test / cucumber properties but do not for main properties - explore

```

13 spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQLDialect
14 #spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalN
15
Run: EmployeeServiceApplication
Hibernate: insert into employee (age,dept_name,email_id,empname,salary) values (?, ?, ?, ?, ?)
Hibernate: insert into employee (age,dept_name,email_id,empname,salary) values (?, ?, ?, ?, ?)
Hibernate: insert into employee (age,dept_name,email_id,empname,salary) values (?, ?, ?, ?, ?)
Hibernate: insert into employee (age,dept_name,email_id,empname,salary) values (?, ?, ?, ?, ?)

```

Every DB has dialect property its use is fine tune the query on behalf of developer



A screenshot of the DBeaver database tool interface. On the left, the Database Navigator shows a tree structure of databases, tables, and columns for a schema named 'javatechie'. In the center, a SQL script editor contains the query `select * FROM employee e`. Below it, a results grid displays the following data:

#id	age	dept_name	email_id	empname	salary
6	31	Sales	sarah@example.com	Sarah Lee	53,000
7	29	Research	christopher@example.com	Christopher Clark	54,000
8	34	Development	amanda@example.com	Amanda Martinez	57,000
9	27	Customer Service	James@example.com	James Taylor	51,000
10	36	Quality Assurance	laura@example.com	Laura Rodriguez	59,000
11	31	HR	john.smith@example.com	John Smith	51,000
12	36	Finance	emily.johnson@example.com	Emily Johnson	62,000
13	33	IT	david.brown@example.com	David Brown	56,000

Camel case properties are created with underscore separator column names  
Example property deptName = dept\_name

## 62. Why is it important to configure a physical naming strategy?

```

14 spring.jpa.hibernate.naming.physical-strategy=org.hibernate.boot.model.naming.PhysicalN

```

```
14 } .physical-strategy=org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
15 }
```

The screenshot shows a database query results window. The query is: "select \* FROM employee". The results are displayed in a table with columns: id, age, depName, emailId, EMPNAME, and salary. There are two rows of data:

	id	age	depName	emailId	EMPNAME	salary
1	1	30	HR	john@example.com	John Doe	50,000
2	2	35	Finance	jane@example.com	Jane Smith	60,000

If you do not want hibernate to create column names and we want java property as column name then use physical-strategy as shown below. Now see the column name.

### 63. What are the key benefits of using Spring Data JPA ?

```
1 package com.javatechie.repository;
2
3 import ...
4
5 public interface EmployeeRepository extends JpaRepository<Employee, Integer> {
6
7 }
```

This Employee Repository will perform all CURD operations on the entity Employee. It reduced Boiler plate code. We have direct methods to play with repository

```
1 package com.javatechie.service;
2
3 import org.springframework.data.repository.CrudRepository;
4 import org.springframework.data.repository.query.Param;
5 import org.springframework.stereotype.Service;
6
7 @Service
8 public class EmployeeService {
9
10     private EmployeeRepository repository;
11
12     public Employee saveEmployee(Employee employee) {
13         repository.find
14             .findAll(Sort sort)
15             .findAll(Pageable pageable)
16             .findAll(Example<S> example)
17             .findAll(Example<S>, Sort sort)
18             .findAll(Example<S>, Pageable pageable)
19             .findById(Iterable<Integer> ids)
20             .findBy(Example<S> example, Function<FetchableFluentQuery<...>, R)
21             .findById(Integer id)
22             .findOne(Example<S> example)
23     }
24 }
```

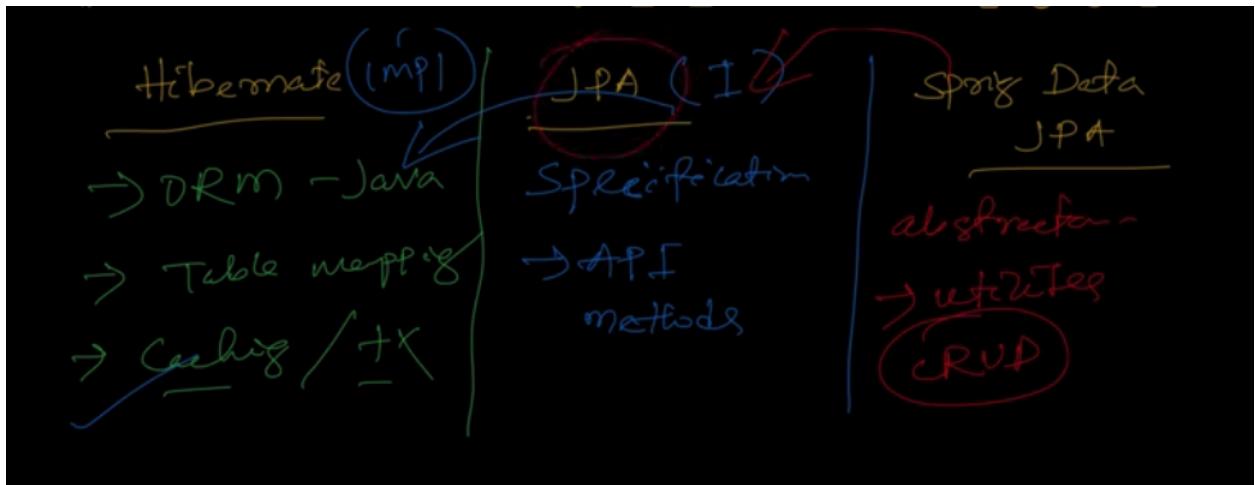
The screenshot shows a Java code editor with the following code:

```
import ...  
  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
  
    //get all the employee from the given deptName  
    |  
    List<Employee> findByDeptName(String deptName);  
  
    List<Employee> findBySalaryAndAge(double salary, int age);  
  
}
```

The code is part of a Spring Data JPA repository interface named `EmployeeRepository`. It includes two custom query methods: `findByDeptName` and `findBySalaryAndAge`.

We do not need to write a query in the code. We can cover the query into methods very easily.

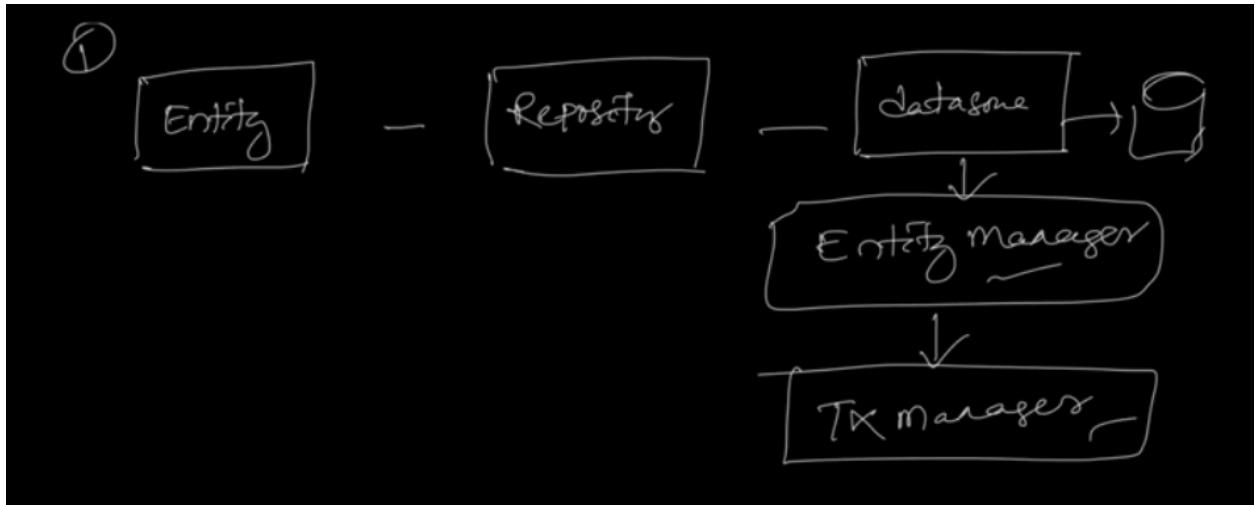
#### 64. What are the differences between Hibernate, JPA, and Spring Data JPA?



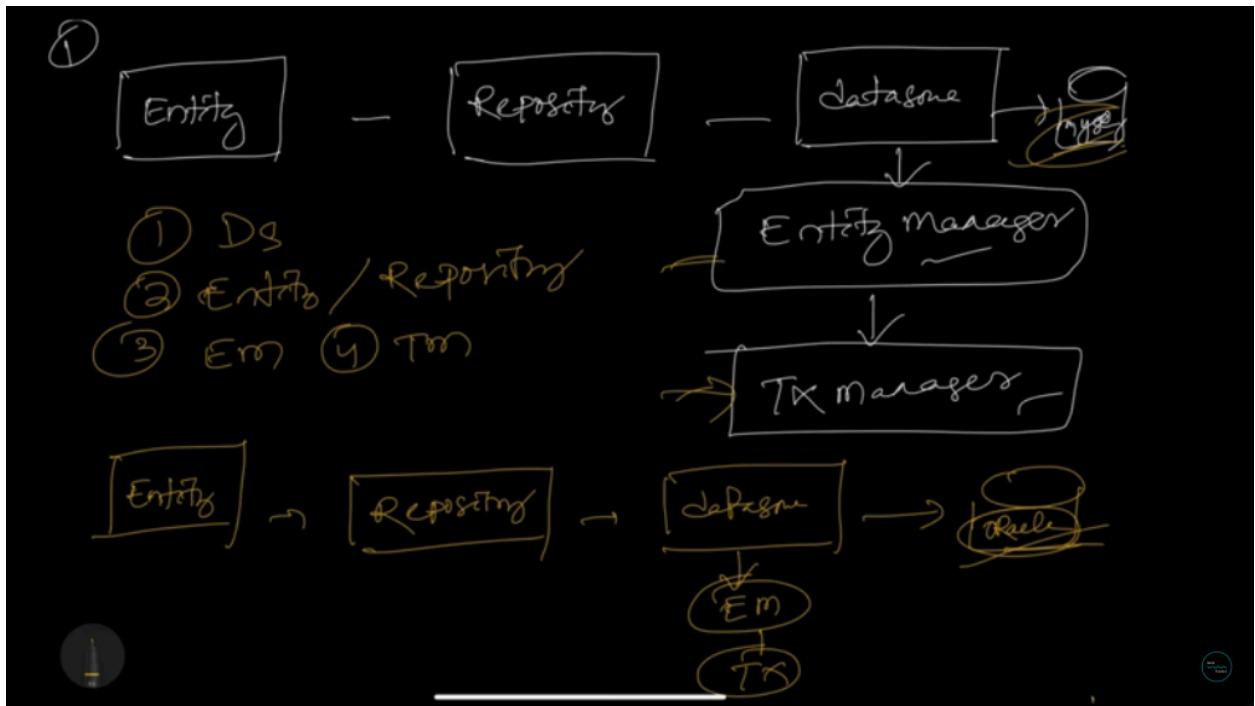
- Hibernate is ORM Tool For Java, it takes care of java to table mapping and caching and transaction.
- JPA is Interface / specification. Hibernate is implementation class of JPA interface
- Spring Data JPA is another level of abstraction on JPA which allows us to perform CRUD operations and custom methods like `findByDeptName` / `findBySalaryAndAge`

## 65. How can you connect multiple databases or data sources in a single application?

Oracle table for payments and sql table for orders. How you connect them.



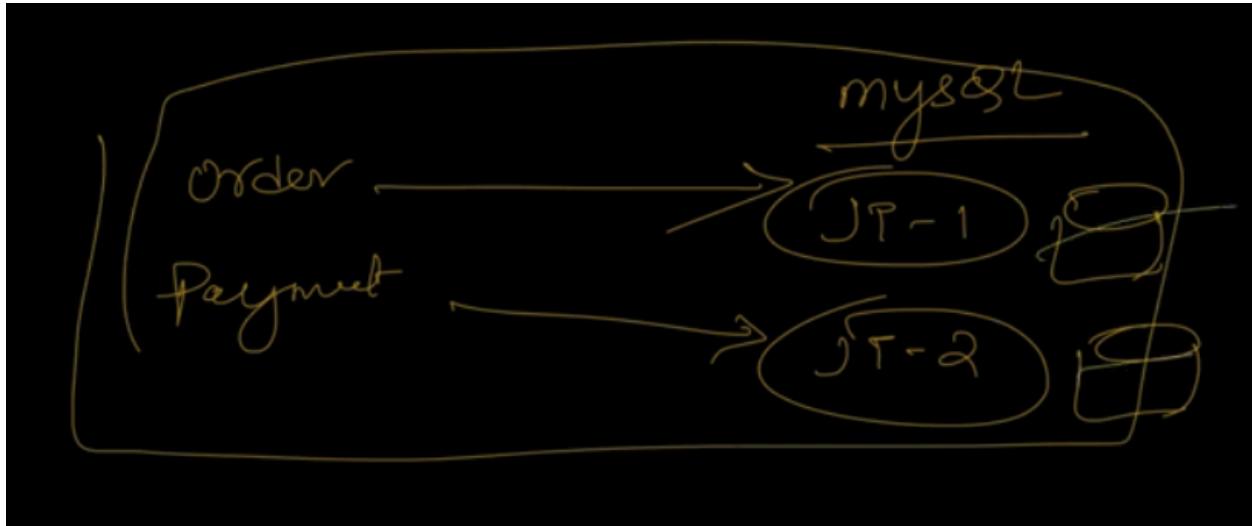
So far we have created an entity, a repository and datasource for one DB. Datasource will create entity manager and transaction manager.



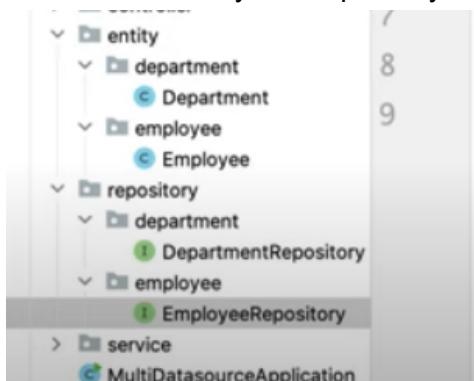
Similarly we have to create an entity, repository and data source for the other DB. Datasource will create entity manager and transaction manager.

Because we are using different DBs we have to do customization for entity manager and transaction manager to handle separately.

Let us see how this can be performed. But tutor used only SQL with different schemas to explain this concept



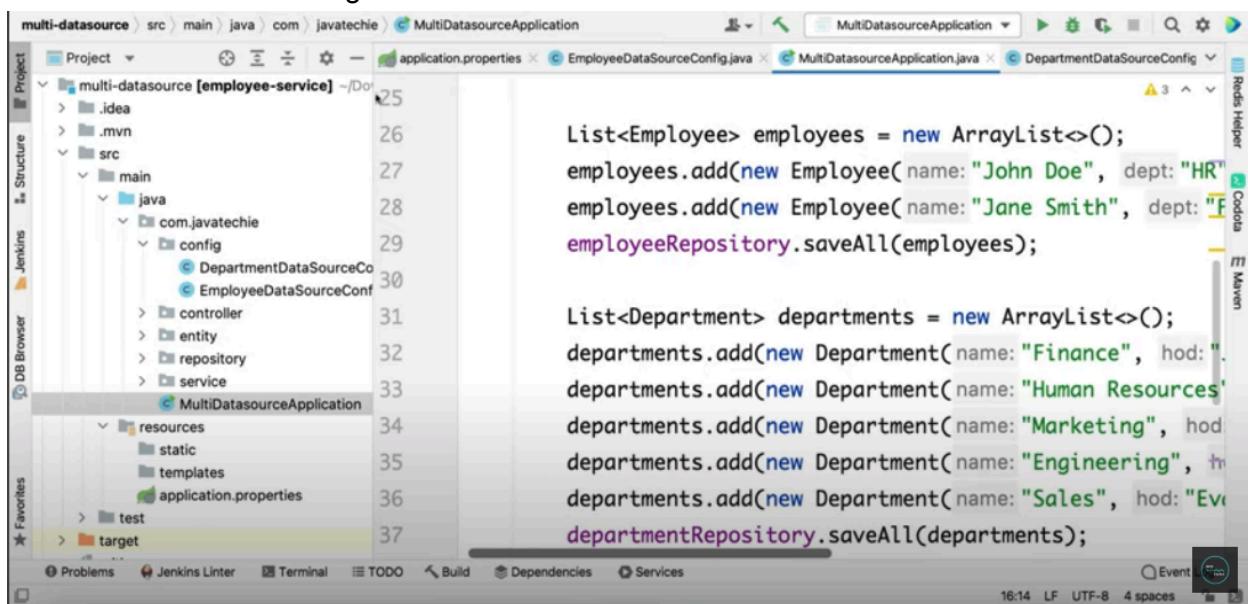
We have two entity and repository and data source



```
3 #EMPLOYEE DATASOURCE PROPERTIES
4 spring.datasource.employee.driver-class-name=com.mysql.cj.jdbc.Driver
5 spring.datasource.employee.url = jdbc:mysql://localhost:3306/javatechie_ds1
6 spring.datasource.employee.username = root
7 spring.datasource.employee.password = Password

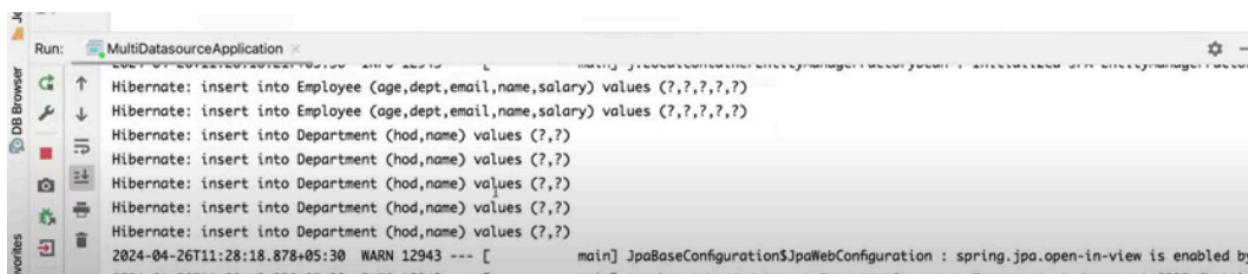
8
9 #DEPARTMENT DATASOURCE PROPERTIES
10 spring.datasource.department.driver-class-name=com.mysql.cj.jdbc.Driver
11 spring.datasource.department.url = jdbc:mysql://localhost:3306/javatechie_ds2
12 spring.datasource.department.username = root
13 spring.datasource.department.password = Password
14
```

## Let us create custom configuration



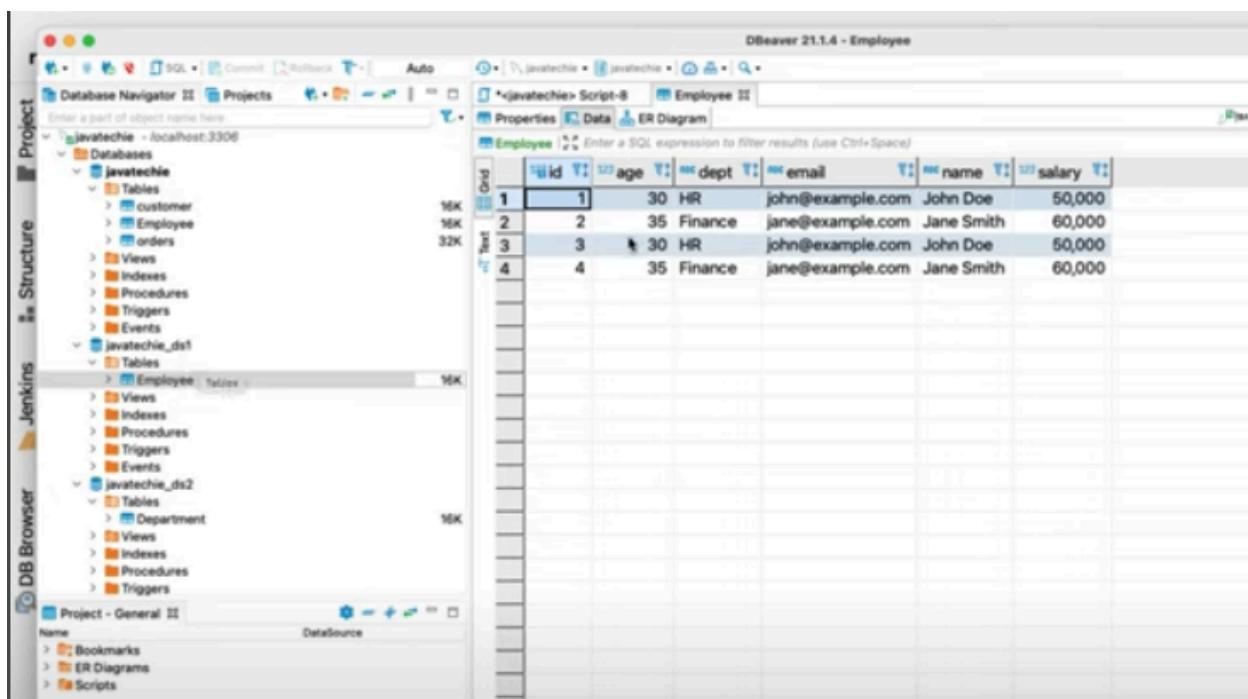
The screenshot shows an IDE interface with the following details:

- Project Structure:** Shows the project tree for "multi-datasource [employee-service]". It includes a "src" folder containing "main", "java", "com.javatechie", "config", "controller", "entity", "repository", "service", and "MultiDataSourceApplication".
- Code Editor:** Displays Java code for "MultiDataSourceApplication.java". The code uses Spring Data JPA annotations to manage two databases: "EmployeeDataSourceConfig" and "DepartmentDataSourceConfig". It defines repositories for "Employee" and "Department" entities and saves them to their respective databases.
- Terminal:** Shows the command "Hibernate: insert into Employee (age,dept,email,name,salary) values (?, ?, ?, ?, ?)" repeated multiple times, indicating successful database insertions.
- Logs:** Shows a log entry from "main" class: "WARN 12943 --- [main] JpaBaseConfiguration\$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default for Hibernate".



The screenshot shows an IDE interface with the following details:

- Project Structure:** Shows the project tree for "multi-datasource [employee-service]". It includes a "src" folder containing "main", "java", "com.javatechie", "config", "controller", "entity", "repository", "service", and "MultiDataSourceApplication".
- Code Editor:** Displays Java code for "MultiDataSourceApplication.java". The code uses Spring Data JPA annotations to manage two databases: "EmployeeDataSourceConfig" and "DepartmentDataSourceConfig". It defines repositories for "Employee" and "Department" entities and saves them to their respective databases.
- Terminal:** Shows the command "Hibernate: insert into Employee (age,dept,email,name,salary) values (?, ?, ?, ?, ?)" repeated multiple times, indicating successful database insertions.
- Logs:** Shows a log entry from "main" class: "WARN 12943 --- [main] JpaBaseConfiguration\$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default for Hibernate".



The screenshot shows the DB Browser for MySQL interface with the following details:

- Database Navigator:** Shows the database structure for "javatechie" and "javatechie\_ds1".
- Data View:** Shows the "Employee" table data. The table has columns: id, age, dept, email, name, and salary. The data is as follows:

	id	age	dept	email	name	salary
1	1	30	HR	john@example.com	John Doe	50,000
2	2	35	Finance	jane@example.com	Jane Smith	60,000
3	3	30	HR	john@example.com	John Doe	50,000
4	4	35	Finance	jane@example.com	Jane Smith	60,000

The screenshot shows the DBBeaver interface with the title "DBBeaver 21.1.4 - Department". The left sidebar displays the "Database Navigator" with a tree view of databases, tables, and objects. The main area shows the "Properties" tab for the "Department" table, which has 10 rows of data:

	ID	HOD	Name
1	1	John Doe	Finance
2	2	Jane Smith	Human Resources
3	3	Alice Johnson	Marketing
4	4	Bob Williams	Engineering
5	5	Eva Brown	Sales
6	6	John Doe	Finance
7	7	Jane Smith	Human Resources
8	8	Alice Johnson	Marketing
9	9	Bob Williams	Engineering
10	10	Eva Brown	Sales

Whenever we have multiple databases we have make one as primary/default

```
multi-datasource / src / main / java / com / javatechie / config / DepartmentDataSourceConfig.java

Code Blame 60 lines (51 loc) - 2.62 KB
15
16     import javax.sql.DataSource;
17     import java.util.HashMap;
18     import java.util.Map;
19
20     @Configuration
21     @EnableTransactionManagement
22     @EnableRepositories{
23         entityManagerFactoryRef = "departmentEntityManagerFactory",
24         transactionManagerRef = "departmentTransactionManager",
25         basePackages = { "com.javatechie.repository.department" })
26     public class DepartmentDataSourceConfig {
27
28         @Bean(name = "departmentProperties")
29         @ConfigurationProperties("spring.datasource.department")
30         public DataSourceProperties dataSourceProperties() {
31             return new DataSourceProperties();
32         }
33
34         @Bean(name = "departmentDataSource")
35         @ConfigurationProperties(prefix = "spring.datasource.department")
36         public DataSource dataSource(@Qualifier("departmentProperties") DataSourceProperties properties) {
37             return properties.initializeDataSourceBuilder().build();
38         }
39
40
41         @Bean(name = "departmentEntityManagerFactory")
42         public LocalContainerEntityManagerFactoryBean entityManagerFactoryBean(
43             EntityManagerFactoryBuilder builder, @Qualifier("departmentDataSource") DataSource dataSource) {
44             Map<String, Object> jpaProps = new HashMap<>();
45             jpaProps.put("hibernate.hbm2ddl.auto", "update");
46             jpaProps.put("hibernate.dialect", "org.hibernate.dialect.MySQLDialect");
47             return builder.dataSource(dataSource)
48                 .properties(jpaProps)
49                 .packages("com.javatechie.entity.department")
50                 .persistenceUnit("department").build();
51         }
52
53         @Bean(name = "departmentTransactionManager")
54         @ConfigurationProperties("spring.jpa")
55         public PlatformTransactionManager transactionManager(
56             @Qualifier("departmentEntityManagerFactory") EntityManagerFactory entityManagerFactory) {
57             return new JpaTransactionManager(entityManagerFactory);
58         }
59     }
60 }
```

## 66. What are the different ways to define custom queries in Spring Data JPA ?

This can be done in three different ways

Using method syntax / JPQL (Java Persistence Query Language) / SQL

```
14     //1. method syntax
15     //2. QUERY (JPQL)
16     //3. SQL
17     //select * from Employee where salary=?//SQL
18     //select * from Employee e where e.salary=?//JPQL
19
20     @Query(value = "SELECT e from Employee e WHERE e.salary > :salary")
21     List<Employee> findEmployeeWithJPQL(@Param("salary") double salary);
22
23     @Query(value = "SELECT * from Employee where salary >?1",nativeQuery = true)
24     List<Employee> findEmployeeWithSQL( double salary);
25
26     List<Employee> findBySalaryGreaterThan(double salary);
```

For JPQL Query and SQL - method names can be anything. For Method Syntax, the syntax name should have a specific format as shown in the above screen. Method syntax internally uses JPQL during execution.

```
46
47     public void deleteEmployee(int id) { repository.deleteById(id); }
48
49
50
51
52
53
54     public List<Employee> filterBySalary(double salary) {
55         log.info("Results with JPQL {}",repository.findEmployeeWithJPQL(salary));
56         log.info("Results with NATIVE SQL {}",repository.findEmployeeWithSQL(salary));
57         return repository.findBySalaryGreaterThan(salary);
58     }
```

Create Control for this.

```
43
44
45     @GetMapping("/filterBySalary")
46     public List<Employee> filterBySalary(@RequestParam double salary) { return employeeService.filterBySalary(salary); }
47
48
```

```

[{"id": 2, "name": "Jane Smith", "deptName": "Finance", "salary": 60000, "emailId": "jane@example.com", "age": 35}, {"id": 5, "name": "Michael Wilson", "deptName": "Operations", "salary": 58000, "emailId": "michael@example.com", "age": 33}, {"id": 8, "name": "Amanda Martinez", "deptName": "Development", "salary": 57000, "emailId": "amanda@example.com", "age": 34}]

```

```

EmployeeServiceApplication x
2024-04-26T11:42:12.955+05:30 INFO 14688 --- [nio-9191-exec-1] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring root WebApplicationContext
2024-04-26T11:42:12.957+05:30 INFO 14688 --- [nio-9191-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-04-26T11:42:12.960+05:30 INFO 14688 --- [nio-9191-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms
Hibernate: select e1_0.id,e1_0.age,e1_0.deptName,e1_0.emsailId,e1_0.EMPNAME,e1_0.salary from Employee e1_0 where e1_0.salary>?
2024-04-26T11:42:13.192+05:30 INFO 14688 --- [nio-9191-exec-1] com.javatechie.service.EmployeeService : Results with JPQL
Hibernate: SELECT * from Employee where salary >?
2024-04-26T11:42:13.262+05:30 INFO 14688 --- [nio-9191-exec-1] com.javatechie.service.EmployeeService : Results with NATIVE
Hibernate: select e1_0.id,e1_0.age,e1_0.deptName,e1_0.emsailId,e1_0.EMPNAME,e1_0.salary from Employee e1_0 where e1_0.salary>?

```

If you keenly notice the logs for method and JPQL the queries are the same. For sql the query is different.

67. Can you provide example to find results based on some range

Ex 1 : Find list of employees whose age in between 30 lakh to 35 lakh

Ex 2 : find avg salary of employees in an organization

In the repository class

```
28
29     List<Employee> findByAgeBetween(int min, int max);
30
31     // findBySalaryAvg
32     // avgSalary();
33
34     @Query(value = "SELECT AVG(e.salary) FROM Employee e")
35     double avgSalary();
36
37 }
```

The screenshot shows a code editor with Java code. The code defines a repository interface with two methods: `findByAgeBetween` and `avgSalary`. The `avgSalary` method is annotated with `@Query` and contains a query string. The code editor has a sidebar with icons for Jenkins, DB Browser, and favorites.

In the service class

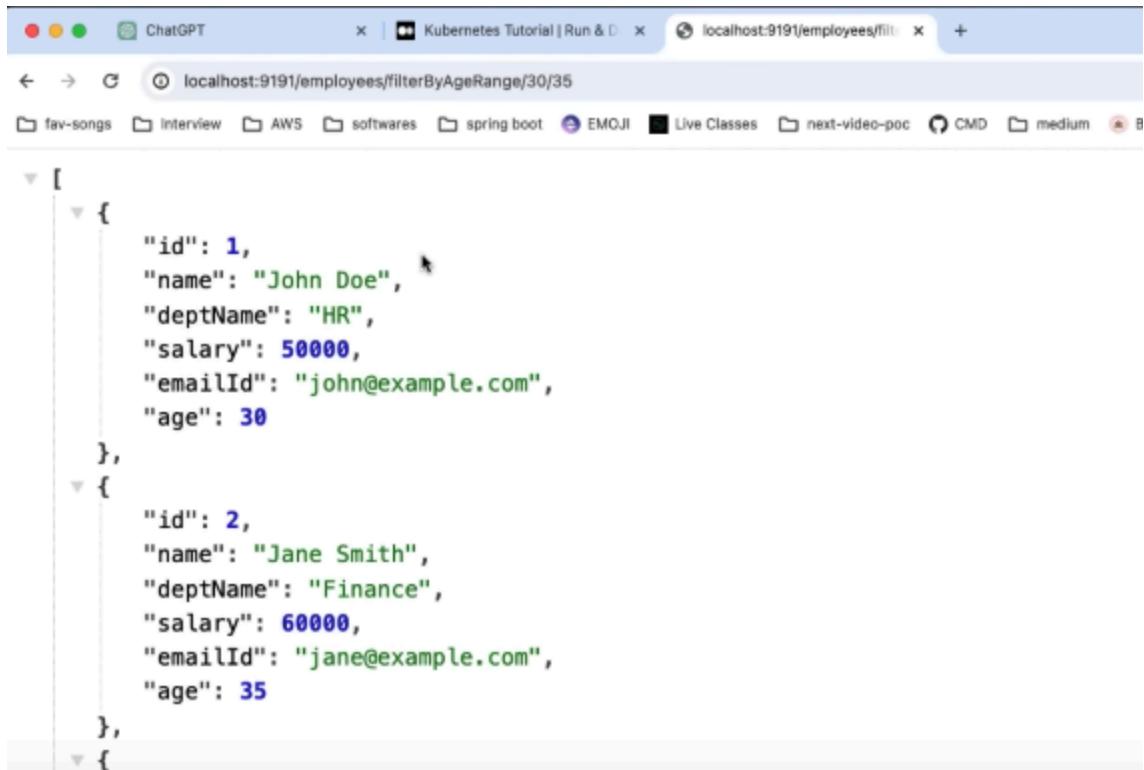
```
59
60     public List<Employee> findEmployeesByAgeRange(int minAge, int maxAge) {
61         return repository.findByAgeBetween(minAge, maxAge);
62     }
63
64     public Optional<Double> getAverageSalary() {
65         return repository.avgSalary();
66     }
67 }
```

The screenshot shows a code editor with Java code. The code defines a service class with two methods: `findEmployeesByAgeRange` and `getAverageSalary`. The `getAverageSalary` method calls the `avgSalary` method from the repository. The code editor has a sidebar with icons for Jenkins, DB Browser, and favorites.

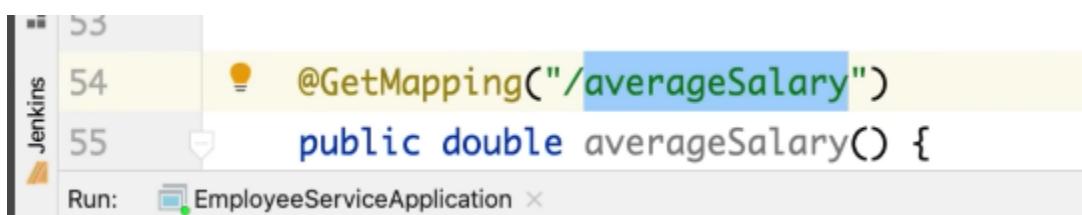
Controller

```
49     @GetMapping("/filterEmployeesByAgeRange/{minAge}/{maxAge}")
50     public List<Employee> filterEmployeesByAgeRange(@PathVariable int minAge, @PathVariable int maxAge) {
51         return employeeService.findEmployeesByAgeRange(minAge, maxAge);
52     }
53 }
```

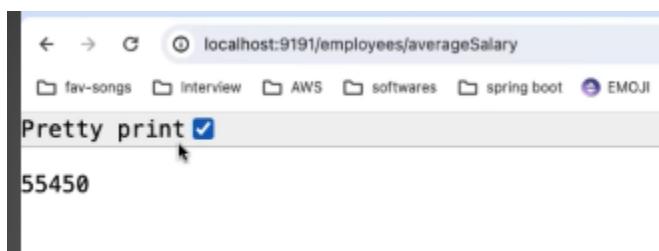
The screenshot shows a code editor with Java code. The code defines a controller with a single endpoint `/filterEmployeesByAgeRange` mapped to the `filterEmployeesByAgeRange` method. This method takes `minAge` and `maxAge` as path variables and returns a list of employees. The code editor has a sidebar with icons for Jenkins, DB Browser, and favorites.



```
[{"id": 1, "name": "John Doe", "deptName": "HR", "salary": 50000, "emailId": "john@example.com", "age": 30}, {"id": 2, "name": "Jane Smith", "deptName": "Finance", "salary": 60000, "emailId": "jane@example.com", "age": 35}]
```



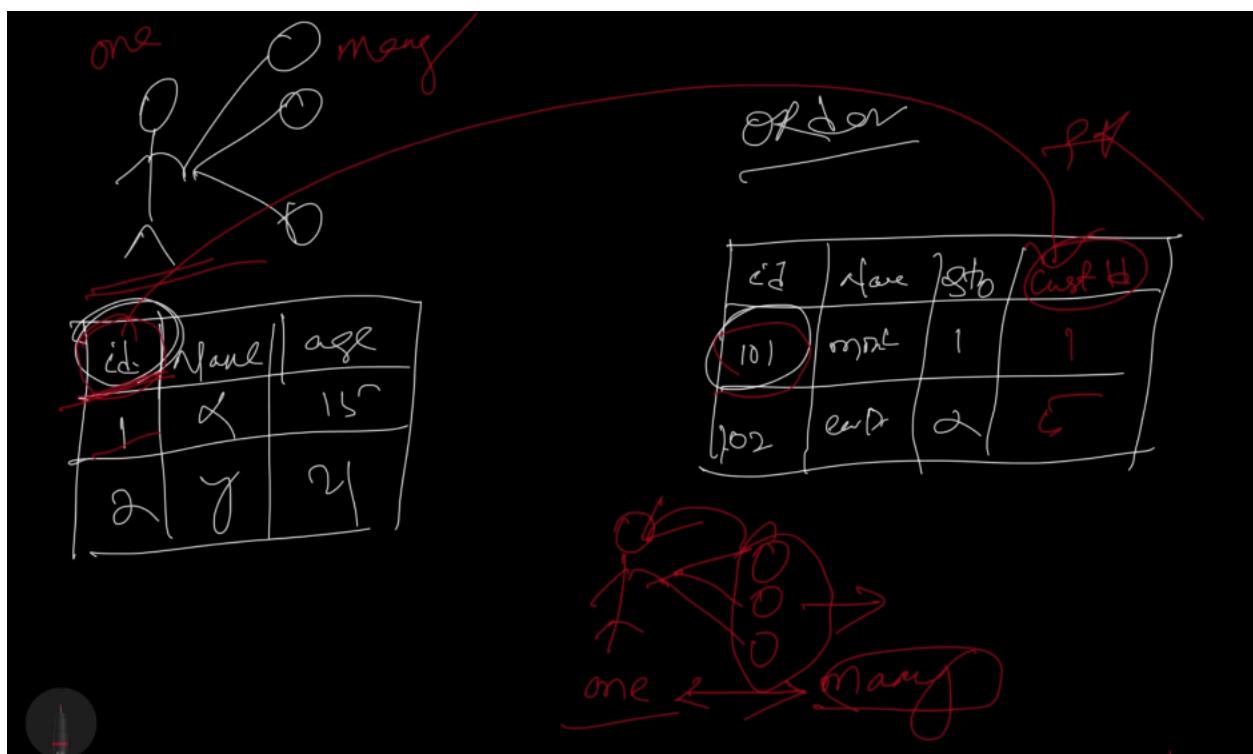
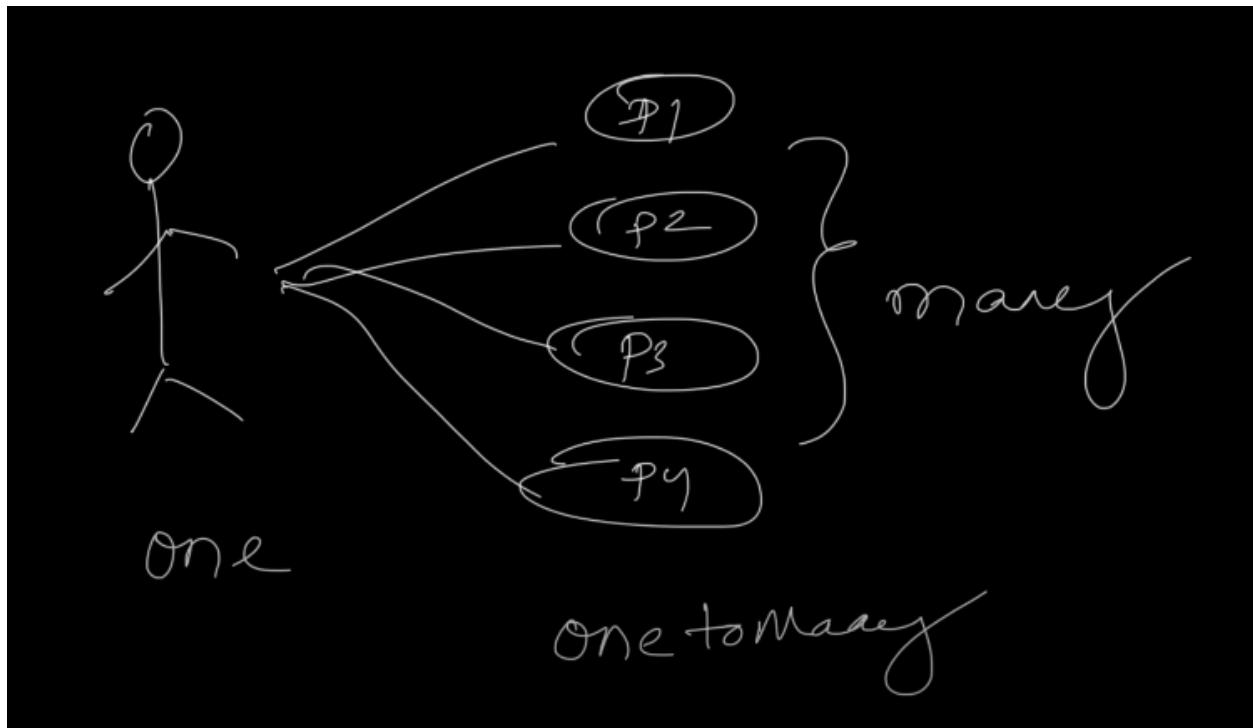
```
53
54     @GetMapping("/averageSalary")
55     public double averageSalary() {
```



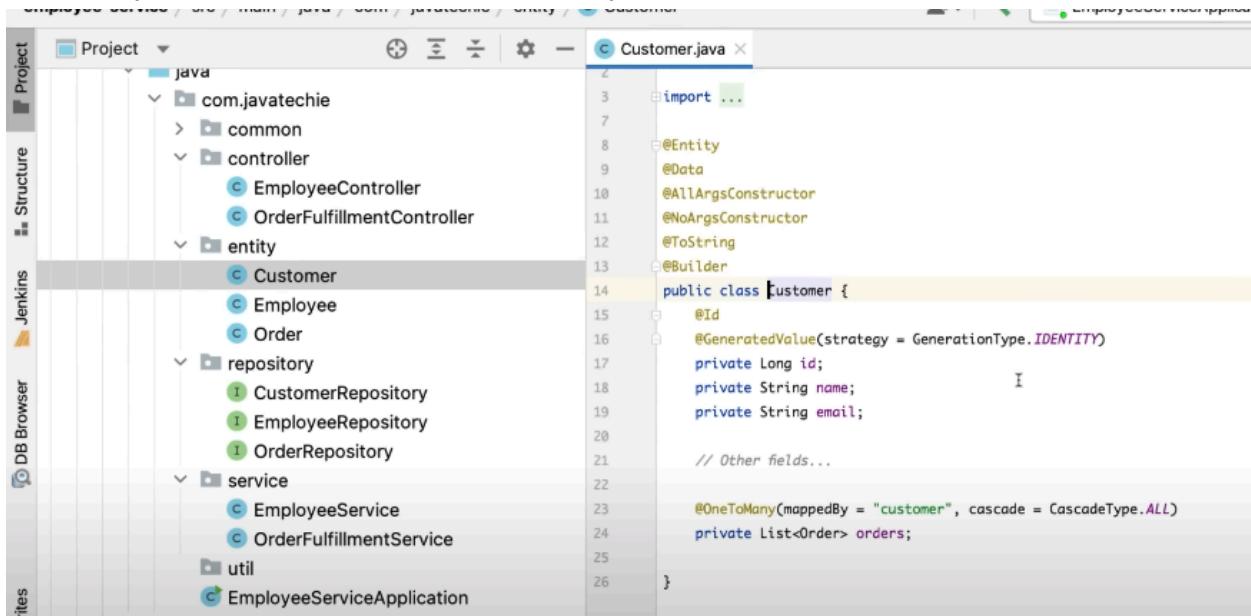
```
localhost:9191/employees/averageSalary
Pretty print 
55450
```

68. How will you define entity relationships or association mapping in Spring Data JPA?

(Use Case: Consider an e-commerce application where each customer can have multiple orders. Here, a Customer entity can be associated with multiple Order entities.)

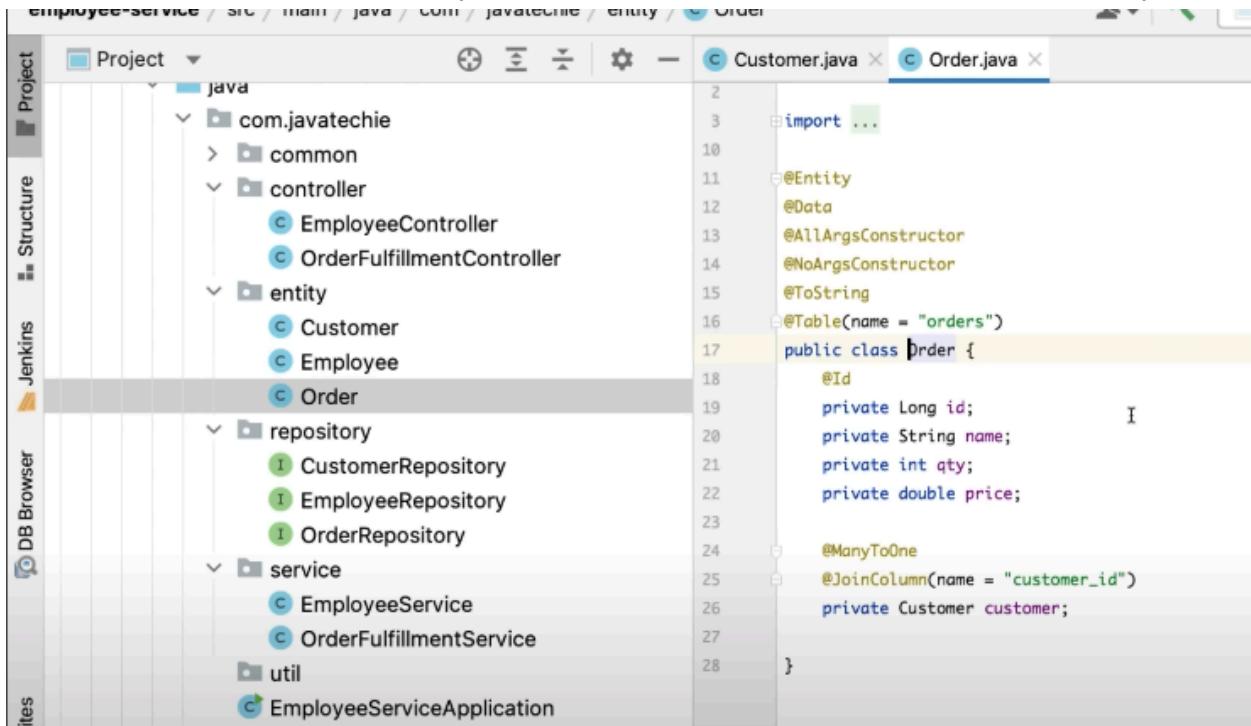


Notice line number 23 in customer entity where we specified relationship with order entity  
CascadeType all will insert data into order entity.



```
Customer.java
import ...
@Entity
@Data
@Builder
public class Customer {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
    // Other fields...
    @OneToMany(mappedBy = "customer", cascade = CascadeType.ALL)
    private List<Order> orders;
}
```

Notice line number 24 in order entity where we specified relationship with customer entity



```
Order.java
import ...
@Entity
@Table(name = "orders")
public class Order {
    @Id
    private Long id;
    private String name;
    private int qty;
    private double price;
    @ManyToOne
    @JoinColumn(name = "customer_id")
    private Customer customer;
}
```

And we have corresponding two repositories

Go to file

src

- main
- java/com/javatechie
- common
- controller
- entity
  - Customer.java
  - Employee.java
  - Order.java
- repository
  - CustomerRepository.java
  - EmployeeRepository.java
  - OrderRepository.java
- service
- EmployeeServiceApplication.j...

resources

test/java/com/javatechie

README.md

pom.xml

**Code** Blame 28 lines (14 loc) · 753 Bytes

```
1 package com.javatechie.repository;
2
3 import com.javatechie.common.CustomerOrderDTO;
4 import com.javatechie.entity.Customer;
5 import com.javatechie.entity.Employee;
6 import org.springframework.data.jpa.repository.JpaRepository;
7 import org.springframework.data.jpa.repository.Query;
8
9 import java.util.List;
10
11 public interface CustomerRepository extends JpaRepository<Customer, Integer> {
12
13     //join query
14     @Query(value = "SELECT c.name , COUNT(o) FROM Customer c JOIN c.orders o GROUP BY c.id")
15     List<Object> findCustomerOrderCount();
16
17     @Query(value = "SELECT NEW com.javatechie.common.CustomerOrderDTO(c.name , COUNT(o), SUM(o.price)) FROM Customer c JOIN c.orders o GROUP BY c.id")
18     List<CustomerOrderDTO> findCustomerOrderCountResponse();
19
20 }
```

**Code** Blame 13 lines (6 loc) · 253 Bytes

```
1 package com.javatechie.repository;
2
3 import com.javatechie.entity.Employee;
4 import com.javatechie.entity.Order;
5 import org.springframework.data.jpa.repository.JpaRepository;
6
7 public interface OrderRepository extends JpaRepository<Order, Integer> {
```

```
12  @Service
13  public class OrderFulfillmentService {
14
15      @Autowired
16      private CustomerRepository customerRepository;
17
18      @Autowired
19      private OrderRepository orderRepository;
20
21      @
22      public Customer createOrder(OrderRequest orderRequest) {
23          Customer customer= (Customer) orderRequest.getCustomer();
24          customer.getOrders().forEach(c->c.setCustomer(customer));
25          return customerRepository.save(customer);
26
27
28      public List<Object[]> findCustomerOrderCount() { return null; }
29
30      public List<CustomerOrderDTO> findCustomerOrderCountResponse() { return null; }
31
32
33      }
34
35  }
```

The highlighted area is for many to one logic

```
12  @
13  public Customer createOrder(OrderRequest orderRequest) {
14
15      Customer customer= (Customer) orderRequest.getCustomer();
16
17      customer.getOrders().forEach(c->c.setCustomer(customer));
18
19      return customerRepository.save(customer);
20
21  }
```

POST http://localhost:9191/e ● GET http://localhost:8181/ap ● POST http://localhost:7070/g ● + No environment

HTTP http://localhost:9191/ecom/addOrder

POST http://localhost:9191/ecom/addOrder

Save Send

Params Authorization Headers (10) Body Pre-request Script Tests Settings Cookies Beautify

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2     "customer": {
3         "name": "peter",
4         "email": "peter@gmail.com",
5         "orders": [
6             {
7                 "id": 287,
8                 "name": "Order 137312",
9                 "qty": 2,
10                "price": 1000.0
11            },
12        ]
13    }
14 }
```

Body Cookies Headers (5) Test Results 200 OK 45 ms 74.52 KB Save as example

Home Workspaces API Network DBeaver 21.1.4 - <javatechie> Script-B

Database Navigator Projects SQL Commit Rollback Auto

Enter a part of object name here

javatechie - localhost:3306

Databases

javatechie

Tables

customer Employee orders

Views Indexes Procedures Triggers Events

javatechie\_ds1

Tables

Employee Views Indexes Procedures Triggers Events

javatechie\_ds2

Tables

Department Views Indexes Procedures Triggers

Project - General

DataSource

select \* FROM customer c

customer 1

select \* FROM customer c Enter a SQL expression to filter results (use Ctrl+Space)

Grid	id	email	name
1	1	johnsmith@gmail.com	john smith
2	2	peter@gmail.com	peter

16K  
16K  
32K

16K

orders 1 23

select \* FROM orders o Enter a SQL expression to filter results (use Ctrl+Space)

	id	name	price	qty	customer_id
1	112	Order 112	25	2	1
2	186	Order 186	30	1	1
3	273	Order 273	20	3	1
4	287	Order 137312	1,000	2	2
5	298	Order 298	20	3	1
6	432	Order 24e3	12,020	3	2
7	28,372	Order 4t746	30,000	1	2

69. Is this possible to execute Join query in Spring Data JPA ? If yes, how can you add some insights ?

As per tutor it is not recommended - because each micro service has its own database. However in the following way we can address above question

```
6K  
2K //customer name , count of product
```

We want customer name from customer entity and count of product from order table for that customer

```
11 public interface CustomerRepository extends JpaRepository<Customer, Integer> {
12
13     //join query
14     @Query(value = "SELECT c.name , COUNT(o) FROM Customer c JOIN c.orders o GROUP BY c.id")
15     List<Object[]> findCustomerOrderCount();
16 }
```

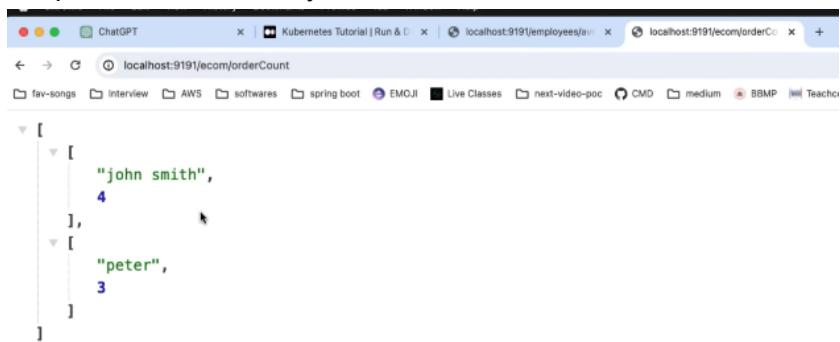
### Service class

```
28     public List<Object[]> findCustomerOrderCount() {  
29         return customerRepository.findCustomerOrderCount();  
30     }
```

### Controller class

```
12     @RestController  
13     @RequestMapping("/ecom")  
14     public class OrderFulfillmentController {  
15  
16         @Autowired  
17         private OrderFulfillmentService orderFulfillmentService;  
18  
19         @PostMapping("/addOrder")  
20         public void addOrder(@RequestBody Order order) {  
21             orderFulfillmentService.addOrder(order);  
22         }  
23         @GetMapping("/orderCount")  
24         public List<Object[]> getCustomerOrderCount() {  
25             return orderFulfillmentService.findCustomerOrderCount();  
26         }  
27     }
```

### Response came in array format



The screenshot shows a browser window with the URL `localhost:9191/ecom/orderCount`. The page displays a JSON array of two elements, each containing a name and a count. The first element is "john smith" with a count of 4, and the second is "peter" with a count of 3.

```
[{"name": "john smith", "count": 4}, {"name": "peter", "count": 3}]
```

Create DTO to get it as an object

The screenshot shows a Java code editor with the file `CustomerOrderDTO.java` open. The code defines a class `CustomerOrderDTO` with three private fields: `customerName`, `orderCount`, and `price`. The code is annotated with `@AllArgsConstructor` and `@NoArgsConstructor`.

```
employee-service > src > main > java > com > javatechie > common > c CustomerOrderDTO
Customer.java X CustomerRepository.java X CustomerOrderDTO.java X Order.java X
Project Jenkins DB Browser
8 @AllArgsConstructor
9 @NoArgsConstructor
10 public class CustomerOrderDTO {
11     private String customerName;
12     private Long orderCount;
13     private double price;
14
15 }
16
```

Direct mapping from query to DTO provided below.

The screenshot shows a Java code editor with the file `OrderFulfillmentService.java` open. It contains two methods: `findCustomerOrderCount()` which returns a list of objects, and `findCustomerOrderCountResponse()` which returns a list of `CustomerOrderDTO`. Both methods delegate to the `customerRepository`.

```
@Query(value = "SELECT NEW com.javatechie.common.CustomerOrderDTO(c.name , COUNT(o), SUM(o.price)) FROM Customer c JOIN c.orders o GROUP BY c.id")
List<CustomerOrderDTO> findCustomerOrderCountResponse();

main > java > com > javatechie > service > OrderFulfillmentService > m findCustomerOrderCountResponse EmployeeServiceApplication
PROJECT Customer.java X CustomerRepository.java X CustomerOrderDTO.java X Order.java X OrderFulfillmentController.java X OrderFulfillmentService.java X Employee
28     public List<Object[]> findCustomerOrderCount() {
29         return customerRepository.findCustomerOrderCount();
30     }
31
32     public List<CustomerOrderDTO> findCustomerOrderCountResponse() {
33         return customerRepository.findCustomerOrderCountResponse();
34     }
35
36 }
```

```

28
29     @GetMapping("/orderCount/response")
30     public List<CustomerOrderDTO> getCustomerOrderCountResponse() {
31         return orderFulfillmentService.findCustomerOrderCountResponse();
32     }
33

```

localhost:9191/ecom/orderCount/response

```

[{"customerName": "john smith", "orderCount": 4, "price": 95}, {"customerName": "peter", "orderCount": 3, "price": 43020}]

```

Below is the log with JPQL query

```

17     @Query(value = "SELECT NEW com.javatechie.common.CustomerOrderDTO(c.name , COUNT(o1.id), sum(o1.price)) FROM Customer c JOIN orders o1 ON c.id=o1.customer_id GROUP BY c.id")
18     List<CustomerOrderDTO> findCustomerOrderCountResponse();
19
Run: EmployeeServiceApplication
2024-04-26T12:34:52.836+05:30 INFO 21621 --- [nio-9191-exec-1] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring root WebApplicationContext
2024-04-26T12:34:52.836+05:30 INFO 21621 --- [nio-9191-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2024-04-26T12:34:52.838+05:30 INFO 21621 --- [nio-9191-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 3 ms
Hibernate: select c1_0.name,count(o1_0.id),sum(o1_0.price) from Customer c1_0 join orders o1_0 on c1_0.id=o1_0.customer_id group by c1_0.id

```

## 70. How will you implement pagination & Sorting in Spring Data JPA ?

sorting

Pagination

sorting and pagination

```
EmployeeService.java
67
68     public List<Employee> findEmployeesWithSorting(String field){
69         return repository.findAll(Sort.by(Sort.Direction.ASC,field));
70     }
71
72     public Page<Employee> findEmployeesWithPagination(int pageNumber,int pageSize){
73         return repository.findAll(PageRequest.of(pageNumber, pageSize));
74         //1000
75         //pageSize : 1-100, 101-200 , 201-300
76         //offset : 0,1,2,3,4,
77         //20
78         // pageSize=5 / 10 (0,1)
79         //pageNumber : 0 (0-5) , 1 (6-10) , 2 (11-15) , 3 (16-20)
```

```
EmployeeService.java
83
84     public Page<Employee> findEmployeesWithPaginationAndSorting(int pageNumber, int pageSize) {
85         return repository.findAll(PageRequest.of(pageNumber, pageSize).withSort(Sort.by(field)));
86     }
```

```
    andSorting(int pageNumber, int pageSize, String field){
        number, pageSize).withSort(Sort.by(field)));
```

## Controllers

```
Controller.java
61     @GetMapping("/sort")
62     public List<Employee> findEmployeesWithSorting(@RequestParam String field) {
63         return employeeService.findEmployeesWithSorting(field);
64     }
65
66     @GetMapping("/page")
67     public Page<Employee> findEmployeesWithPagination(@RequestParam int offset, @RequestParam int pageSize) {
68         return employeeService.findEmployeesWithPagination(offset, pageSize);
69     }
```

Sort by name

Sort by age

Sort by department

Any sorting we can perform

```
localhost:9191/employees/sort?field=name

[{"id": 8, "name": "Amanda Martinez", "deptName": "Development", "salary": 57000, "emailId": "amanda@example.com", "age": 34}, {"id": 18, "name": "Amanda Taylor", "deptName": "Sales", "salary": 58000, "emailId": "amanda.taylor@example.com", "age": 35}, {"id": 10, "name": "Laura Rodriguez", "deptName": "Quality Assurance", "salary": 59000, "emailId": "laura@example.com", "age": 36}]
```

```
localhost:9191/employees/sort?field=age

[{"id": 8, "name": "Amanda Martinez", "deptName": "Development", "salary": 57000, "emailId": "amanda@example.com", "age": 34}, {"id": 18, "name": "Amanda Taylor", "deptName": "Sales", "salary": 58000, "emailId": "amanda.taylor@example.com", "age": 35}, {"id": 10, "name": "Laura Rodriguez", "deptName": "Quality Assurance", "salary": 59000, "emailId": "laura@example.com", "age": 36}]
```

## Pagination

1st page with size 5

```
localhost:9191/employees/page?offset=0&pageSize=5

{
    "id": 3,
    "name": "David Johnson",
    "deptName": "IT",
    "salary": 55000,
    "emailId": "david@example.com",
    "age": 32
},
{
    "id": 4,
    "name": "Emily Brown",
    "deptName": "Marketing",
    "salary": 52000,
    "emailId": "emily@example.com"
}
```

## Pagination and sorting

```
localhost:9191/employees/pageAndSort?offset=1&pageSize=10&field=age

{
    "id": 1,
    "name": "Michael Lee",
    "deptName": "Operations",
    "salary": 59000,
    "emailId": "michael.lee@example.com",
    "age": 34
},
{
    "id": 2,
    "name": "Jane Smith",
    "deptName": "Finance",
    "salary": 60000,
    "emailId": "jane@example.com",
    "age": 35
},
{
    "id": 18,
```