

## 1. Explain the steps to establish MySQL connectivity in a project.

### MySQL Connectivity Steps

1. **Add Dependencies** – Include MySQL and Spring Boot Starter JPA in `pom.xml`.
2. **Configure Database** – Set up MySQL credentials in `application.properties`.
3. **Create Entity Class** – Define JPA entity with annotations.
4. **Create Repository** – Extend `JpaRepository` to handle database operations.
5. **Use Service Layer** – Implement business logic using the repository.
6. **Run Application** – Spring Boot auto-configures and connects to MySQL.

## 2. Explain the role of the application.properties file in MySQL connectivity.

### Role of `application.properties` in MySQL Connectivity:

- Stores database configuration (URL, username, password, driver).
- Enables Hibernate auto-configuration.
- Defines properties like `spring.datasource.url`, `spring.datasource.username`, etc.

## 3. What is JPA Auditing?

JPA Auditing automatically tracks entity creation and modification timestamps. It helps in maintaining record logs.

## 4. Explain JPA Auditing using @CreatedDate and @LastModifiedDate.

**@CreatedDate** – Captures timestamp when an entity is first created.

**@LastModifiedDate** – Updates timestamp when an entity is modified.

**Requires @EnableJpaAuditing** in the main class and an entity with **@EntityListeners(AuditingEntityListener.class)**.

## Q.2

**Build a CRUD Rest API Project using the MySQL Database and JPA concept on Customer Entity created in the previous assignment.**

1. **Create a CustomerController** to handle CRUD operations for the Customer entity.
2. **Create a CustomerService** to manage business logic and interact with the CustomerRepository.
3. **In the CustomerRepository interface**, which should extend JpaRepository, implement the following query methods:
  - **Find Customers by First Name**
  - **Find Customers by Last Name**
  - **Find Customers by Email**

- Find Customers by Salary Range
  - Find Customers with Salary Greater Than a Specific Value
  - Find Customers by Name Starting With
  - Find Customers by Email Domain
4. Test the Query Methods using Postman

**B]**

With reference to the Customer entity class created in the previous assignments, apply @CreateDate and @LastModifiedDate to your Customer entity class.

**Hint:**

Apply in the following way:

@CreateDate

private Instant CreatedAt;

@LastModifiedDate

private Instant ModifiedAt;

**Entity:**

```
Customer.java x CustomerApplication/pom.xml application.properties application.properties Cu
#
1 package com.amex.CustomerApplication.entity;
2
3 import java.time.Instant;
4
5 import org.springframework.data.annotation.CreatedDate;
6 import org.springframework.data.annotation.LastModifiedDate;
7
8 import jakarta.persistence.Entity;
9 import jakarta.persistence.GeneratedValue;
10 import jakarta.persistence.GenerationType;
11 import jakarta.persistence.Id;
12 import lombok.AllArgsConstructor;
13 import lombok.Data;
14 import lombok.NoArgsConstructor;
15
16 @Data
17 @NoArgsConstructor
18 @AllArgsConstructor
19 @Entity
20 public class Customer {
21
22     @Id
23     @GeneratedValue(strategy = GenerationType.IDENTITY)
24     private Integer customerId;
25     private String firstName;
26     private String lastName;
27     private String email;
28     private String address;
29
30     @CreateDate
31     private Instant createdAt;
32
33     @LastModifiedDate
34     private Instant modifiedAt;
35 }
36
```

## Repository:

```
1 package com.amex.CustomerApplication.repository;
2
3 import com.amex.CustomerApplication.entity.Customer;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 import java.util.List;
7
8 public interface CustomerRepository extends JpaRepository<Customer, Integer> {
9     List<Customer> findByFirstName(String firstName);
10    List<Customer> findByLastName(String lastName);
11    List<Customer> findByEmail(String email);
12    List<Customer> findBySalaryBetween(Double minSalary, Double maxSalary);
13    List<Customer> findBySalaryGreaterThan(Double salary);
14    List<Customer> findByFirstNameStartingWith(String prefix);
15    List<Customer> findByEmailEndingWith(String domain);
16 }
17
```

## Service:

```
1 package com.amex.CustomerApplication.services;
2
3 import com.amex.CustomerApplication.entity.Customer;
4
5 @Service
6 public class CustomerService {
7
8     @Autowired
9     private CustomerRepository customerRepository;
10
11     public List<Customer> getAllCustomers() {
12         return customerRepository.findAll();
13     }
14
15     public Optional<Customer> getCustomerById(Integer id) {
16         return customerRepository.findById(id);
17     }
18
19     public Customer createCustomer(Customer customer) {
20         return customerRepository.save(customer);
21     }
22
23     public Customer updateCustomer(Customer customer) {
24         return customerRepository.save(customer);
25     }
26
27     public void deleteCustomer(Integer id) {
28         customerRepository.deleteById(id);
29     }
30
31     public List<Customer> findCustomersByFirstName(String firstName) {
32         return customerRepository.findByFirstName(firstName);
33     }
34
35     public List<Customer> findCustomersByLastName(String lastName) {
36         return customerRepository.findByLastName(lastName);
37     }
38
39     public List<Customer> findCustomersByEmail(String email) {
40         return customerRepository.findByEmail(email);
41     }
42
43     public List<Customer> findCustomersBySalaryRange(Double minSalary, Double maxSalary) {
44         return customerRepository.findBySalaryBetween(minSalary, maxSalary);
45     }
46
47     public List<Customer> findCustomersBySalaryGreaterThan(Double salary) {
48         return customerRepository.findBySalaryGreaterThan(salary);
49     }
50
51     public List<Customer> findCustomersByNameStartingWith(String prefix) {
52         return customerRepository.findByFirstNameStartingWith(prefix);
53     }
54
55     public List<Customer> findCustomersByEmailDomain(String domain) {
56         return customerRepository.findByEmailEndingWith(domain);
57     }
58 }
59
```

## Controller:

```
Customer.java  CustomerApplication/pom.xml  CustomerController.java ×  application.properties  application.prop
10
11 @RestController
12 @RequestMapping("/customers")
13 public class CustomerController {
14
15     @Autowired
16     private CustomerService customerService;
17
18     @GetMapping
19     public List<Customer> getAllCustomers() {
20         return customerService.getAllCustomers();
21     }
22
23     @GetMapping("/{id}")
24     public Optional<Customer> getCustomerById(@PathVariable Integer id) {
25         return customerService.getCustomerById(id);
26     }
27
28     @PostMapping
29     public Customer createCustomer(@RequestBody Customer customer) {
30         return customerService.createCustomer(customer);
31     }
32
33     @PutMapping("/{id}")
34     public Customer updateCustomer(@PathVariable Integer id, @RequestBody Customer customer) {
35         customer.setCustomerId(id);
36         return customerService.updateCustomer(customer);
37     }
38
39     @DeleteMapping("/{id}")
40     public void deleteCustomer(@PathVariable Integer id) {
41         customerService.deleteCustomer(id);
42     }
43
44     @GetMapping("/search/firstName")
45     public List<Customer> findCustomersByFirstName(@RequestParam String firstName) {
46         return customerService.findCustomersByFirstName(firstName);
47     }
48
49     @GetMapping("/search/lastName")
50     public List<Customer> findCustomersByLastName(@RequestParam String lastName) {
51         return customerService.findCustomersByLastName(lastName);
52     }
53 }
```

**Database:**

