**Employee.java:**

@Entity

@DynamicUpdate // Hibernate-specific: updates only changed columns

@SelectBeforeUpdate // Checks if update is necessary

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

@Column(nullable = false)

private String name;

@Column

private double salary;

@CreationTimestamp // Hibernate-specific

private LocalDateTime createdAt;

@UpdateTimestamp // Hibernate-specific

private LocalDateTime updatedAt;

// Getters and Setters

}

**application.properties:**# DataSource

spring.datasource.url=jdbc:mysql://localhost:3306/employee\_db

spring.datasource.username=root

spring.datasource.password=root

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# Hibernate JPA Config

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.format\_sql=true

# Performance Tuning

spring.jpa.properties.hibernate.jdbc.batch\_size=30

spring.jpa.properties.hibernate.order\_inserts=true

spring.jpa.properties.hibernate.order\_updates=true

spring.jpa.properties.hibernate.generate\_statistics=true

**EmployeeService.java:**

@Service

public class EmployeeService {

@PersistenceContext

private EntityManager entityManager;

@Transactional

public void saveEmployeesInBatch(List<Employee> employees) {

int batchSize = 30;

for (int i = 0; i < employees.size(); i++) {

entityManager.persist(employees.get(i));

if (i > 0 && i % batchSize == 0) {

entityManager.flush();

entityManager.clear();

}

}

entityManager.flush();

entityManager.clear();

}

}

**EmployeeController.java:**

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

@PostMapping("/batch")

public ResponseEntity<String> saveBatch(@RequestBody List<Employee> employees) {

employeeService.saveEmployeesInBatch(employees);

return ResponseEntity.ok("Batch saved successfully");

}

}