Exercise 1: Employee Management System - Overview and Setup

1. **Creating a Spring Boot Project**

```
curl https://start.spring.io/starter.zip -d dependencies=data-jpa,h2,web,lombok -d
baseDir=EmployeeManagementSystem -o EmployeeManagementSystem.zip
unzip EmployeeManagementSystem.zip
cd EmployeeManagementSystem
```

2. **Configuring Application Properties**:

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Exercise 2: Employee Management System - Creating Entities

1. **Creating JPA Entities**:

Employee.java

```
@Entity
@Table(name = "employees")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Employee {
   @Id
   @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   private String name;
   private String email;
   @ManyToOne
   private Department department;
}
```

Department.java

```
@Entity
@Table(name = "departments")
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Department {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    @OneToMany(mappedBy = "department")
    private List<Employee> employees;
}
```

2. **Mapping Entities to Database Tables**:

- @Entity and @Table annotations map the classes to database tables.
- @Id and @GeneratedValue annotations define the primary key and auto-generation strategy.
- @ManyToOne and @OneToMany annotations define the one-to-many relationship between Department and Employee.

Exercise 3: Employee Management System - Creating Repositories

1. **Creating Repositories**:
   EmployeeRepository.java

```
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    List<Employee> findByName(String name);
    List<Employee> findByEmail(String email);
}
```

DepartmentRepository.java

```
public interface DepartmentRepository extends JpaRepository<Department, Long> {
    Department findByName(String name);
}
```

Exercise 4: Employee Management System - Implementing CRUD Operations

1. **Basic CRUD Operations**:
   EmployeeController.java

```
@RestController
@RequestMapping("/employees")
public class EmployeeController {
    @Autowired
    private EmployeeRepository employeeRepository;

    @GetMapping
    public List<Employee> getAllEmployees() {
        return employeeRepository.findAll();
    }

    @GetMapping("/{id}")
    public Employee getEmployeeById(@PathVariable Long id) {
        return employeeRepository.findById(id).orElseThrow();
    }

    @PostMapping
    public Employee createEmployee(@RequestBody Employee employee) {
        return employeeRepository.save(employee);
    }

    @PutMapping("/{id}")
    public Employee updateEmployee(@PathVariable Long id, @RequestBody
Employee employee) {
        Employee existingEmployee = employeeRepository.findById(id).orElseThrow();
        existingEmployee.setName(employee.getName());
        existingEmployee.setEmail(employee.getEmail());
        existingEmployee.setDepartment(employee.getDepartment());
        return employeeRepository.save(existingEmployee);
    }

    @DeleteMapping("/{id}")
    public void deleteEmployee(@PathVariable Long id) {
        employeeRepository.deleteById(id);
    }
}
```

Exercise 5: Employee Management System - Defining Query Methods

1. **Defining Query Methods:**
   EmployeeRepository.java

```
public interface EmployeeRepository extends JpaRepository<Employee,
Long> {
    List<Employee> findByName(String name);
    List<Employee> findByEmail(String email);
    List<Employee> findByDepartment(Department department);
}
```

   DepartmentRepository.java

```
public interface DepartmentRepository extends JpaRepository<Department,
Long> {
    Department findByName(String name);
}
```

2. **Named Queries:**
   Employee.java

```
@Entity
@Table(name = "employees")
@NamedQuery(name = "Employee.findByNameAndEmail", query =
"SELECT e FROM Employee e WHERE e.name = ?1 AND e.email = ?2")
public class Employee {
    // ...
}
```

   EmployeeRepository.java

```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee,
Long> {
    @Query(name = "Employee.findByNameAndEmail")
    List<Employee> findByNameAndEmail(String name, String email);
}
```

Exercise 6: Employee Management System - Implementing Pagination and Sorting

1. **Pagination**:
   EmployeeController.java

```
@GetMapping
public Page<Employee> getAllEmployees(Pageable pageable) {
    return employeeRepository.findAll(pageable);
}
```

2. **Sorting:**
   EmployeeController.java

```
@GetMapping
public Page<Employee> getAllEmployees(Pageable pageable) {
    return employeeRepository.findAll(pageable);
}

@GetMapping("/sorted")
public List<Employee> getAllEmployeesSorted(@RequestParam String sortField,
@RequestParam Direction sortDirection) {
    return employeeRepository.findAll(Sort.by(sortDirection, sortField));
}
```

Exercise 7: Employee Management System - Enabling Entity Auditing

1. **Entity Auditing**
   Employee.java

```
@Entity
@Table(name = "employees")
@EntityListeners(AuditingEntityListener.class)
public class Employee {
    // ...
    @CreatedBy
    private String createdBy;
    @LastModifiedBy
    private String lastModifiedBy;
    @CreatedDate
    private LocalDateTime createdDate;
    @LastModifiedDate
    private LocalDateTime lastModifiedDate;
}
```

   AuditingConfig.java

```
@Configuration
@EnableJpaAuditing
public class AuditingConfig {
    @Bean
    AuditorAware<String> auditorProvider() {
        return () -> Optional.of("system");
    }
}
```

Exercise 8: Employee Management System - Creating Projections

1. **Projections:**
   EmployeeProjection.java

```
public interface EmployeeProjection {
    Long getId();
    String getName();
    String getEmail();
    DepartmentProjection getDepartment();
    interface DepartmentProjection {
        Long getId();
        String getName();
    }
}
```

EmployeeRepository.java

```
@Repository
public interface EmployeeRepository extends JpaRepository<Employee, Long> {
    List<EmployeeProjection> findProjectedBy();
}
```

Exercise 9: Employee Management System - Customizing Data Source Configuration

1. **Externalizing Configuration**:
   application.properties

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.datasource.secondary.url=jdbc:h2:mem:testdb2
spring.datasource.secondary.driverClassName=org.h2.Driver
spring.datasource.secondary.username=sa
spring.datasource.secondary.password=password
spring.jpa.secondary.database-platform=org.hibernate.dialect.H2Dialect
```

Exercise 10: Employee Management System - Hibernate-Specific Features

1. **Hibernate-Specific Annotations:**

   Employee.java

```
@Entity
@Table(name = "employees")
public class Employee {
    // ...
    @Type(type = "json")
    @Column(columnDefinition = "json")
    private Map<String, Object> metadata;
}
```

2. **Configuring Hibernate Dialect and Properties**:

application.properties

```
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.H2Dialect
spring.jpa.properties.hibernate.format_sql=true
spring.jpa.properties.hibernate.show_sql=true
```

3. **Batch Processing**:

   EmployeeService.java

```
@Transactional
public void bulkInsert(List<Employee> employees) {
    int batchSize = 10;
    for (int i = 0; i < employees.size(); i++) {
        employeeRepository.save(employees.get(i));
        if (i > 0 && i % batchSize == 0) {
            employeeRepository.flush();
            employeeRepository.clear();
        }
    }
}
```