**Objectives**

* **Demonstrate implementation of Query Methods feature of Spring Data JPA**
  + **Query Methods - Search by containing text, sorting, filter with starting text, fetch between dates, greater than or lesser than, top**
    - **Query methods - https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#jpa.query-methods.query-creation**

**Spring Data JPA - Query Methods Feature**

Spring Data JPA supports defining custom queries through method names in the repository interface. These are known as query methods, and Spring generates the required SQL based on naming conventions.

**Common Query Method Patterns**

**Text Matching**

* findByNameContaining(String keyword)  
  Finds records where the name contains the given text.
* findByNameStartingWith(String prefix)  
  Retrieves records where name starts with the specified prefix.
* findByNameEndingWith(String suffix)  
  Retrieves records where name ends with the given suffix.

**Date-Based Filters**

* findByCreatedDateBetween(Date start, Date end)  
  Gets records created within a specific date range.

**Numeric Comparisons**

* findBySalaryGreaterThan(Double amount)  
  Fetches records where salary is greater than the specified value.
* findByAgeLessThan(Integer age)  
  Finds records where age is less than the given value.

**Top or First N Records**

* findTop3ByOrderByScoreDesc()  
  Retrieves the top 3 records ordered by score in descending order.
* findFirstByOrderByNameAsc()  
  Retrieves the first record ordered by name in ascending order.

**Sorting**

* findAllByOrderByNameAsc()  
  Returns all records sorted by name in ascending order.
* findAllByOrderByDateDesc()  
  Returns all records sorted by date in descending order.

**How to Use**

* Define these methods in an interface that extends JpaRepository.
* Spring will implement these automatically based on the method name.

Example:

java

CopyEdit

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContaining(String keyword);

List<Country> findByCodeStartingWith(String prefix);

List<Country> findTop5ByOrderByNameAsc();

}

* **Demonstrate implementation of O/R Mapping**
  + **@ManyToOne, @JoinColumn, @OneToMany, FetchType.EAGER, FetchType.LAZY, @ManyToMany, @JoinTable, mappedBy**
    - **Relationships reference - https://www.baeldung.com/spring-data-rest-relationships**
* In Spring Data JPA, Object-Relational Mapping (O/R Mapping) is used to define relationships between different entity classes, reflecting how tables are related in a relational database. These relationships are established using annotations such as @ManyToOne, @OneToMany, @ManyToMany, and are often combined with @JoinColumn, @JoinTable, and mappedBy to control the join behavior and ownership.
* The @ManyToOne annotation represents a many-to-one relationship where multiple child entities are associated with a single parent. It is typically used with @JoinColumn to specify the foreign key in the child table. For example, if many employees belong to one department, the Employee class will use @ManyToOne with @JoinColumn(name="dept\_id") to point to the Department.
* Conversely, the @OneToMany annotation is used in the parent class to map the reverse side of the relationship. This is often accompanied by the mappedBy attribute to indicate the field that owns the relationship. For instance, in the Department class, you can declare @OneToMany(mappedBy="department") to map the list of employees.
* The FetchType enum controls how related entities are loaded. FetchType.EAGER loads the related entity immediately with the parent, while FetchType.LAZY defers loading until explicitly accessed. Lazy loading is the default for @OneToMany and @ManyToMany, whereas eager loading is the default for @ManyToOne.
* For many-to-many relationships, the @ManyToMany annotation is used in both related entities. This is typically paired with @JoinTable, which defines the join table that connects the two entity tables. You can specify join columns and inverse join columns to control the foreign keys.

**Hands on 1**

**Write queries on country table using Query Methods**   
  
Following are the list of queries that is required for an application. Implement these queries using Query Methods feature of Spring Data JPA. Click [**here**](https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#jpa.query-methods.query-creation) for reference. Include appropriate methods in OrmLearnApplication and test the same. 

* An application has a search text box for searching by country. When typing characters on the text box, a list of all the matching countries should be displayed. For example, if 'ou' is entered in the search box the following countries should be displayed. Write a Query Method to achieve this feature. Implement this method in CountryRepository.

BV       Bouvet Island

DJ       Djibouti

GP       Guadeloupe

GS       South Georgia and the South Sandwich Islands

LU       Luxembourg

SS       South Sudan

TF       French Southern Territories

UM       United States Minor Outlying Islands

ZA       South Africa

* Enhance the above method to return the countries in ascending order. Modify the query method name defined in the previous problem to achieve this.

BV       Bouvet Island

DJ       Djibouti

TF       French Southern Territories

GP       Guadeloupe

LU       Luxembourg

ZA       South Africa

GS       South Georgia and the South Sandwich Islands

SS       South Sudan

UM       United States Minor Outlying Islands

* To select a country an alphabet index is displayed in a web page, when the user clicks on the alphabet, all the countries starting that alphabet needs to be displayed. For example if the alphabet choose is 'Z', then the following countries should be displayed. Write a query method to get this feature incorporated.

ZM       Zambia

ZW       Zimbabwe

**Hands on 2**

**Write queries on stock table using Query Methods**   
  
With one year stock data of Facebook, Google and Netflix, we need to implement Spring Data JPA Query Methods for the following scenarios:  
  
**Sample Data**  
Sample data for implementing this hands on is provided to you in the platform

**Setup stock data**

* Create a new table for storing stock details.

CREATE TABLE IF NOT EXISTS `ormlearn`.`stock` (

  `st\_id` INT NOT NULL AUTO\_INCREMENT,

  `st\_code` varchar(10),

  `st\_date` date,

  `st\_open` numeric(10,2),

  `st\_close` numeric(10,2),

  `st\_volume` numeric,

  PRIMARY KEY (`st\_id`)

);

* The file stock-data.csv in spring-data-jpa-files folder contains the stock data of Facebook, Google and Netflix from 18 Oct 2018 to 17 Oct 2019. This is public data downloaded from finance.yahoo.com.
* Open stock-data.csv file in Excel
* Include the following formula in F2 cell, this will display the insert script in F2 cell.

=CONCATENATE("insert into stock (st\_code, st\_date, st\_open, st\_close, st\_volume) values (""", B2, """, """, YEAR(A2), "-", MONTH(A2), "-", DAY(A2), """, ", C2, ", ", D2, ", ", E2, ");")

* Drag the formula for all the rows of data in F column.
* Copy all data in F column and paste it in Notepad or Notepad++ and save it as a file named stock-data.sql. Execute this script in mysql command line client which populates data into ormlearn.stock table. Following command assumes that the git project is available in D:.

mysql> source D:\spring-data-jpa-files\stock-data.sql

* Create new class Stock in orm-learn project and define the required mapping annotations.
* Create StockRepository class to write the Query Methods
* Create methods in OrmLearnApplication to test by autowiring StockRepository directly.

**Query Methods required for the following scenarios**

* Get all stock details of Facebook in the month of September 2019. Expected data result of Query Method below.

+---------+------------+---------+----------+-----------+

| st\_code | st\_date    | st\_open | st\_close | st\_volume |

+---------+------------+---------+----------+-----------+

| FB      | 2019-09-03 |  184.00 |   182.39 |   9779400 |

| FB      | 2019-09-04 |  184.65 |   187.14 |  11308000 |

| FB      | 2019-09-05 |  188.53 |   190.90 |  13876700 |

| FB      | 2019-09-06 |  190.21 |   187.49 |  15226800 |

| FB      | 2019-09-09 |  187.73 |   188.76 |  14722400 |

| FB      | 2019-09-10 |  187.44 |   186.17 |  15455900 |

| FB      | 2019-09-11 |  186.46 |   188.49 |  11761700 |

| FB      | 2019-09-12 |  189.86 |   187.47 |  11419800 |

| FB      | 2019-09-13 |  187.33 |   187.19 |  11441100 |

| FB      | 2019-09-16 |  186.93 |   186.22 |   8444800 |

| FB      | 2019-09-17 |  186.66 |   188.08 |   9671100 |

| FB      | 2019-09-18 |  188.09 |   188.14 |   9681900 |

| FB      | 2019-09-19 |  188.66 |   190.14 |  10392700 |

| FB      | 2019-09-20 |  190.66 |   189.93 |  19934200 |

| FB      | 2019-09-23 |  189.34 |   186.82 |  13327600 |

| FB      | 2019-09-24 |  187.98 |   181.28 |  18546600 |

| FB      | 2019-09-25 |  181.45 |   182.80 |  18068300 |

| FB      | 2019-09-26 |  181.33 |   180.11 |  16083300 |

| FB      | 2019-09-27 |  180.49 |   177.10 |  14656200 |

+---------+------------+---------+----------+-----------+

* Get all google stock details where the stock price was greater than 1250

+---------+------------+---------+----------+-----------+

| st\_code | st\_date    | st\_open | st\_close | st\_volume |

+---------+------------+---------+----------+-----------+

| GOOGL   | 2019-04-22 | 1236.67 |  1253.76 |    954200 |

| GOOGL   | 2019-04-23 | 1256.64 |  1270.59 |   1593400 |

| GOOGL   | 2019-04-24 | 1270.59 |  1260.05 |   1169800 |

| GOOGL   | 2019-04-25 | 1270.30 |  1267.34 |   1567200 |

| GOOGL   | 2019-04-26 | 1273.38 |  1277.42 |   1361400 |

| GOOGL   | 2019-04-29 | 1280.51 |  1296.20 |   3618400 |

| GOOGL   | 2019-10-17 | 1251.40 |  1252.80 |   1047900 |

+---------+------------+---------+----------+-----------+

* Find the top 3 dates which had highest volume of transactions

+---------+------------+---------+----------+-----------+

| st\_code | st\_date    | st\_open | st\_close | st\_volume |

+---------+------------+---------+----------+-----------+

| FB      | 2019-01-31 |  165.60 |   166.69 |  77233600 |

| FB      | 2018-10-31 |  155.00 |   151.79 |  60101300 |

| FB      | 2018-12-19 |  141.21 |   133.24 |  57404900 |

+---------+------------+---------+----------+-----------+

* Identify three dates when Netflix stocks were the lowest

+---------+------------+---------+----------+-----------+

| st\_code | st\_date    | st\_open | st\_close | st\_volume |

+---------+------------+---------+----------+-----------+

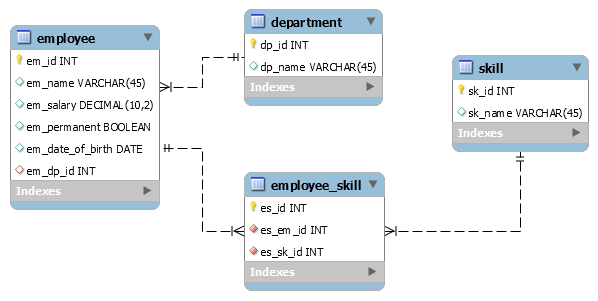
| NFLX    | 2018-12-24 |  242.00 |   233.88 |   9547600 |

| NFLX    | 2018-12-21 |  263.83 |   246.39 |  21397600 |

| NFLX    | 2018-12-26 |  233.92 |   253.67 |  14402700 |

+---------+------------+---------+----------+-----------+

**Hands on 3**

**Create payroll tables and bean mapping**   
  
To demonstrate one to many, many to one and many to many relationships in Hibernate, a schema with entities employee, department and skill will be used. In this hands on we will setup the tables and data, which forms the basis for learning the mappings in Hibernate.  
  
**Schema Structure**  
  
  
  
Follow steps below to create necessary tables:

* Open mysql client in command line
* Use the source command to execute the payroll.sql script file available in spring-data-jpa-files folder. The following command assumes that spring-data-jpa-files folder is in D:.

mysql> source D:\spring-data-jpa-files\payroll.sql

Define bean mapping

* Open orm-learn project in Eclipse
* Create model classes Employee, Department and Skill in com.cognizant.orm-learn.model package
* Define each model should have @Entity and @Table annotations.
* Each id field should be have @Id annotation and @GeneratedValue(strategy = GenerationType.IDENTITY) annotation. @GeneratedValue annotation ensures auto increment of id creation.
* Define @Column against each field.
* Define getters, setters and toString() methods
* Employee
  + private int id;
  + private String name;
  + private double salary;
  + private boolean permanent;
  + private Date dateOfBirth;
* Department
  + private int id;
  + private String name;
* Skill
  + private int id;
  + private String name;
* Create appropriate repository interfaces EmployeeRepository, DepartmentRepository and SkillRepository in repository package

**Hands on 4**

**Implement many to one relationship between Employee and Department**   
  
Follow steps below to defined many to one relationship and perform persistence operations:  
  
**Preparation of Service Classes**

* Create EmployeeService, DepartmentService and SkillService defined with annotation @Service. In each of this class autowire respective repository.
* In each of the service class implement two methods one is to get the entity based on id and the other one is to save the entity. Sample code below provided for EmployeeService, in similar fashion include the methods for DepartmentService and SkillService.
* EmployeeService - get() method

    @Transactional

    public Employee get(int id) {

        LOGGER.info("Start");

        return employeeRepository.findById(id).get();

    }

* EmployeeService - save() method

    @Transactional

    public void save(Employee employee) {

        LOGGER.info("Start");

        employeeRepository.save(employee);

        LOGGER.info("End");

    }

* Include static references of EmployeeService, DepartmentService and SkillService in OrmLearnApplication.
* Assign employeeService, departmentService and skillService from the context in OrmLearnApplication main() method.

**Implementation of @ManyToOne mapping**

* Define department in Employee bean with @ManyToOne and @JoinTable annotation. This defines the relationship between the entities.

    @ManyToOne

    @JoinColumn(name = "em\_dp\_id")

    private Department department;

* Include setters and getters for instance variable department.

**Getting Employee along with Department**

* Create new method testGetEmployee() in OrmLearnApplication
* Implement below code in the method.

    private static void testGetEmployee() {

        LOGGER.info("Start");

        Employee employee = employeeService.get(1);

        LOGGER.debug("Employee:{}", employee);

        LOGGER.debug("Department:{}", employee.getDepartment());

        LOGGER.info("End");

    }

* The above implementation gets the employee with id 1 and displays the employee details and department details.
* Include testGetEmployee() method in main and comment the other test method calls.
* Execute the main method and observe the following:
  + In the logs check the lines where the query is generated.
  + Since the relationship is defined, hibernate fetches department data as well. The query should look something like the below. Observe the department table join in this query. The query is formatted for better readability.

select employee0\_.em\_id as em\_id1\_2\_0\_, employee0\_.em\_date\_of\_birth as em\_date\_2\_2\_0\_,

employee0\_.em\_dp\_id as em\_dp\_id6\_2\_0\_, employee0\_.em\_name as em\_name3\_2\_0\_,

employee0\_.em\_permanent as em\_perma4\_2\_0\_, employee0\_.em\_salary as em\_salar5\_2\_0\_,

  department1\_.dp\_id as dp\_id1\_1\_1\_, department1\_.dp\_name as dp\_name2\_1\_1\_

  from employee employee0\_ left outer join department department1\_

  on employee0\_.em\_dp\_id=department1\_.dp\_id

 where employee0\_.em\_id=?

**NOTE:** SME to explain the learners about Eager Fetch and Lazy Fetch. As per JPA specification by default, Eager Fetch is applied For ManyToOne and OneToOne relationships. Hence department details as well is joined and fetched by Hibernate.  
   
**Add Employee**

* Create new method testAddEmployee() in OrmLearnApplication and implement the following steps
  + Create a new instance of Employee
  + Set the values for the employee using setter method
  + Get a department based on department id 1 using departmentService
  + Set the department in employee based on the department obtained in the previous step
  + Invoke employeeService.save() passing the employee object created
  + Log employee object reference in debug mode
* Include testAddEmployee() invocation in main() method and comment other test methods
* Invoke the main method and check the following:
  + Log should contain select query to get the department and insert statement to add employee
  + Observe that the employee log after save contains the id. Hibernate inserts the records, gets the id and set the id instance variable of employee
  + Check in database if new employee data is inserted in employee table

**Update Employee**

* Create new method testUpdateEmployee() in OrmLearnApplication and implement the following steps
  + Get an employee instance based on employee id using employeeService.get() method
  + Get a department based on department id using departmentService. Use a different department id from the one that is fetched.
  + Set the department in employee based on the department obtained in the previous step
  + Invoke employeeService.save() passing the employee object created
  + Log employee object reference in debug mode
* Include testUpdateEmployee() invocation in main() method and comment other test methods
* Invoke the main method and check the following:
  + Log should contain select query to get the department and employee. There should be update statement that updates employee table
  + Check in database if department id is modified.

**Hands on 5**

**Implement one to many relationship between Employee and Department**   
  
Department.java

* Include new instance variable for set of employees and define the OneToMay annotation

    @OneToMany(mappedBy = "department")

    private Set<Employee> employeeList;

* Include setter and getter for employeeList

OrmLearnApplication.java

* Include new method testGetDepartment()
* In this method, get a department using departmentService.get() passing the id. Select an department id that has more than one employee.
* Log the returned department and department.getEmployeeList()
* Include testGetDepartment() method invocation in main method and comment the other test methods.
* Execute the main() method which will fail with LazyInitializationException. This is because the default fetch type for OneToMany relationship is LAZY, hibernate fetches only department details and does not get the employee details.
* In order to get the employee list as well, modify the annotation to include the fetch type as EAGER. Make this change in employeeList annotation definition of Department class.

    @OneToMany(mappedBy = "department", fetch = FetchType.EAGER)

    private Set<Employee> employeeList;

* After this change try executing the main() method, which will fetch both department and employee

**Hands on 6**

**Implement many to many relationship between Employee and Skill**   
  
**Many to Many mapping defintion**

* Include set of skill list in Employee.java with appropriate getter and setter
* Include set of employee list in Skill.java with appropriate getter and setter
* Include ManyToMany definition in Employee.java as specified below:

    @ManyToMany

    @JoinTable(name = "employee\_skill",

        joinColumns = @JoinColumn(name = "es\_em\_id"),

        inverseJoinColumns = @JoinColumn(name = "es\_sk\_id"))

    private Set<Skill> skillList;

* Include ManyToMany defintion in Skill.java as specified below:

    @ManyToMany(mappedBy = "skillList")

    private Set<Employee> employeeList;

**Fetching Employee along with Skills**

* In testGetEmployee() method of OrmLearnApplication.java include a new line to log employee skill details after the department log.

        LOGGER.debug("Skills:{}", employee.getSkillList());

* Execute the main method which will fail with LazyInitializationException
* Include fetch type as eager in @ManyToMany annotation of Employee.java, which will fetch the skill details as well.

**Add Skill to Employee**

* Include a new method testAddSkillToEmployee() in OrmLearnApplication.java
* Implement the following steps in this method:
  + Identify an employee id and skill id for which a relationship does not exists
  + Get employee based on employee id calling employeeService.get() method
  + Get skill based on skill id calling skillService.get() method
  + Get the skill list from employee and add the skill obtained in the previous step to the skill list
  + Call save method in employeeService passing the employee reference
* Invoke testAddSkillToEmployee() in main method and comment other test methods
* Execute the main method and check employee\_skill table to verify if the skill is added to the employee.