**Spring Data – L1 Assignments**

**Preferred Technologies/Tools to use for implementing solutions:**

1. Use Spring Boot based Maven projects to work on these.
2. Use STS or IntelliJ IDE
3. Use Hibernate as your JPA Provider
4. Use Oracle as your Database

(Above list is the preferred set, you are most welcome to use any IDE/Database/JPA provider of your choice)

**Assignments:**

1. Create a Spring Data application which maintains the below entity.

|  |  |  |
| --- | --- | --- |
| Entity Name | CD |  |
| Properties: |  |  |
|  | cdid | Long |
|  | cdtitle | String |
|  | cdprice | Float |
|  | cdpublisher | String |

Requirements that needs to be implemented in the application:

1. Use import.sql file or add sample records in the respective table.
2. The application should allow me to find all the CD details with all the properties
3. The application should also allow me to query specific CD/a group of CDs either by its cdtitle or cdpublisher
4. Rewrite the above solution of assignment (1) with the below updates/additions:
5. Instead of using import.sql file or add sample records in the respective tables, rewrite the application to include methods to save, update and delete existing records which will allow us to perform the respective operations.
6. Apart from the existing queries, now we would want to use a query that allows me to get ONE CD details by using its ID
7. Rewrite the above solution of assignment (2) with the below updates/additions:
8. Use the appropriate Transactions decorator/annotation (@Transactional) and the ‘readOnly’ property as well if its applicable
9. Use the timeout, rollbackFor transactional properties as well wherever applicable
10. Use the @Modifying annotation wherever applicable
11. Rewrite the solution of assignment (2) with the below updates/additions:
12. Add the below additional queries :
    1. findByPriceRange 🡪 this query should take a range of ‘cdprice’ and return the CD details within that range. (Use query params)
    2. findByPriceRangeNamedParams 🡪 this query should take a range of ‘cdprice’ and return the CD details within that range (use Named query params)
    3. findByTitleMatch 🡪 this query should return the CD based on the title (use the ‘like’ clause)
    4. findByTitleNative🡪 this query should return the CD based on the title (use the ‘native query’)
    5. findAllSorted 🡪 this query should return all the CDs but arranged in the given sorted order (Try 2 sorting scenarios: sort it in the ascending order of cdtitle and sort it in the descending order of cdprice)
13. Rewrite the solution of assignment(2) with the below updates/additions:
14. Add a “Named Query” called ‘findByCdPrice’ which should show all the CDs which have a price greater than the given price.
15. The above query should also show the ‘No. of CDs’ as per the price criteria

(Hint: Implement this as a Named Query i.e. @NamedQuery)

1. Create a Spring Data application which maintains the below entity.

|  |  |  |
| --- | --- | --- |
| Entity Name | Employee |  |
| Properties: |  |  |
|  | empid | Long |
|  | empname | String |
|  | empdoj | Date |
|  | emptotalsalary | Float |
|  | Employeetype | String |

Requirements that needs to be implemented in the application:

1. Implement methods to support adding, modifying and deleting employee entries.
2. In terms of queries, we would need queries that will return all employees, return one employee based on empid and return one/group of employees based on the empdoj.
3. Add queries that can help in querying for a particular employee based on empname but this should allow case-insensitive query

(**Hint:** Use Dynamic query execution methods by using CriteriaQuery, CriteriaBuilder and Predicate class to apply the case-insensitivity filter)

1. Rewrite the above solution of Assignment (6) with the below updates/additions:
2. Remove all existing queries.
3. Add the below queries only:
   1. Query which will just give me a count of all the employees.
   2. Query which will give me a list of all employees who belong to the type “permanent” (other employee type is “contract”)
   3. Query which will give me a list of all employees whose name has a “tha” in them (E.g.: Rujutha, tharini, Supritha etc.,). This should also work in a case-insensitive manner.

(Hint: Implement this as Query by Example. Use the ‘Example’ class, ‘ExampleMatcher’ class and apply the appropriate filter)

1. Use the ‘Employee’ entity that you have already used in the previous assignment as below:

|  |  |  |
| --- | --- | --- |
| Entity Name | Employee |  |
| Properties: |  |  |
|  | empid | Long |
|  | empname | String |
|  | empdoj | Date |
|  | emptotalsalary | Float |
|  | Employeetype | String |

Requirements that needs to be implemented in the application:

1. Add/Save sample Employee records (at least 10 or 20)
2. Create a query that returns all the employees
3. Create a query that returns one employee based on empid
4. Create a query that returns one/many employees based on empname
5. All the above queries should be written in a Type Safe way. (Hint: Use QueryDSL for these queries)
6. Purposely commit some mistakes in the queries written for the previous assignment (8) (a) (b) (c) and see whether you get compile time errors.
7. Create a Spring Data application which maintains the below entity.

|  |  |  |
| --- | --- | --- |
| Entity Name | CD |  |
| Properties: |  |  |
|  | cdid | Long |
|  | cdtitle | String |
|  | cdprice | Float |
|  | cdpublisher | String |

Requirements that needs to be implemented in the application:

1. Create a type safe query to retrieve all the CDs and all the CDs based on a publisher name.

(Hint: Use QueryDSL for these queries)