```
                        Date : 28/01/2021
                        Spring Boot 9AM
                          Mr. RAGHU
                 ---------------------------------
                   Spring Data JPA: Custom Query

@Query("HQL/JPQL") : SELECT/NON-SELECT

*) SELECT:
  all Columns -- List<T>
  One Column  -- List<DT>
  Multiple Col -- List<Object[]>

*) Absract methods inside Repository interface
   provide HQL/JPQL using @Query.

-------------------------------------------------------------
*) Passing Parameters(inputs to Query)
a) Positional Parameters ->  ?position
   Position numbers starts from one(1).
   [Old versions starts from zero, spring data jpa 2.x]

 ... where col operator ?1  .. col operator ?2.. col operator ?3

b) Named Parameters      ->  :name


*) If our Query returns single row data then List is not required
   we can use return type as T, DataType, Object[] but use Object
   (later cast to Object[])

*) In case of Positional Parameter 'param name' can be any name
  but datatype and poisition order must match.


--------------------------Full code-----------------
1. Name : SpringBoot2DataJpaCustomQuery
   Dep : Data Jpa, MySQL, Lombok

2. properties
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/boot9am
spring.datasource.username=root
spring.datasource.password=root

spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

3. Model class
package in.nareshit.raghu.model;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import javax.persistence.Entity;
import javax.persistence.Id;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Employee {
    @Id
    private Integer empId;
    private String empName;
    private Double empSal;
}
```

4. Repository interface

```
package in.nareshit.raghu.repo;

import in.nareshit.raghu.model.Employee;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import java.util.List;

public interface EmployeeRepo extends JpaRepository<Employee,Integer>
{
    @Query("SELECT e FROM in.nareshit.raghu.model.Employee e")
    List<Employee> getAllEmps();

    @Query("SELECT e.empName FROM in.nareshit.raghu.model.Employee e")
    List<String>  getAllEmpNames();

    @Query("SELECT e.empId,e.empName FROM
in.nareshit.raghu.model.Employee e")
    List<Object[]> getAllEmpIdAndNames();

    //------------using parameters----------
    @Query("SELECT e FROM in.nareshit.raghu.model.Employee e WHERE
e.empSal<?1")
    List<Employee> getAllEmpsBySal(Double empSal);

    /*List not required if query returns one row data*/
    @Query("SELECT e FROM in.nareshit.raghu.model.Employee e WHERE
e.empId=?1")
    Employee getOneEmpsById(Integer empId);

    /*List not required if query returns one row data*/
    /*Use Object inplace of Object[] for return type, later down
cast*/
    @Query("SELECT e.empName,e.empSal FROM
in.nareshit.raghu.model.Employee e WHERE e.empId=?1 or e.empName=?2")
    Object getOneEmpNameSalByIdOrName(Integer eid,String ename);
//param name can be any name

    @Query("SELECT e FROM in.nareshit.raghu.model.Employee e ORDER BY
e.empName DESC")
    List<Employee> getAllEmpsSorted();
}
```

5. Runner#1
```java
package in.nareshit.raghu.runner;

import in.nareshit.raghu.model.Employee;
import in.nareshit.raghu.repo.EmployeeRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import java.util.Iterator;
import java.util.List;

//@Component
public class EmployeeInsertRunner implements CommandLineRunner {
    @Autowired
    private EmployeeRepo repo;
    @Override
    public void run(String... args) throws Exception {
        /*repo.save(new Employee(10,"A",2.2));
        repo.save(new Employee(11,"B",3.2));
        repo.save(new Employee(12,"C",4.2));
        */
        //List<Employee> list = repo.getAllEmps();
        //list.forEach(System.out::println);
        //List<String> list = repo.getAllEmpNames();
        //list.forEach(System.out::println);
        List<Object[]> list = repo.getAllEmpIdAndNames();
        //java #8 Stream
        /*list.stream()
                .map(ob->ob[0]+"-"+ob[1])
                .forEach(System.out::println);*/
        Iterator<Object[]> itr = list.iterator();
        while (itr.hasNext()) {
            Object[] ob=itr.next();
            System.out.println(ob[0]+"-"+ob[1]);
        }
        /*for(Object[] ob:list) {
            System.out.println(ob[0]+"-"+ob[1]);
        }*/
    }
}
```

6. Runner#2
```java
package in.nareshit.raghu.runner;

import in.nareshit.raghu.model.Employee;
import in.nareshit.raghu.repo.EmployeeRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import java.util.DoubleSummaryStatistics;
import java.util.List;
import java.util.stream.Collectors;

@Component
```

```
public class EmployeeQueryTestRunner
    implements CommandLineRunner
{
    @Autowired
    private EmployeeRepo repo;

    @Override
    public void run(String... args) throws Exception {
        /*
        List<Employee> list = repo.getAllEmpsBySal(120.0);
        list.forEach(System.out::println);

        //----Java 8 Stream operation----
        list.stream()
                .map(emp-> "Hello Mr/Mrs/Ms."+ emp.getEmpName())
                .forEach(System.out::println);

        Double data = list.stream()

.collect(Collectors.summingDouble(Employee::getEmpSal));
        System.out.println("Full Sal :" + data);

         */
        /*
        Employee emp = repo.getOneEmpsById(10);
        System.out.println(emp);

         */
        /*
        Object emp = repo.getOneEmpNameSalByIdOrName(10,"A");
        System.out.println(emp);
        Object[] ob=(Object[])emp;
        System.out.println(ob[0]+"-"+ob[1]);

         */


        //Sort data using Java 8. Stream
       /* repo.getAllEmps()
                .stream()
                .sorted((e1,e2)-
>e2.getEmpId().compareTo(e1.getEmpId()))
                .forEach(System.out::println);
        */
        repo.getAllEmpsSorted().forEach(System.out::println);

    }
}
```

----

*) If Query gets modified with params, then it may effect position
   numbers in query and params order in method.

```
Ex(old query)
select * from emptab where esal>?1 and eid<?2
List<T> m1(Double a,Integer b)

Ex(modified query)
```

```
select * from emptab where esal>?1 or ename=?2 and eid<?3
List<T> m1(Double a,String m,Integer b)


*) Insted of using numbers use names ie 'Named Parameters'
   Syntax :name

Positional Param :
a. SELECT e FROM Employee WHERE e.empId=?1
b. SELECT e FROM Employee WHERE e.empId=?1 and e.empName=?2

Named  Param :
a. SELECT e FROM Employee WHERE e.empId=:eid
List m1(Integer eid) ; //param name and named parameter must match

b. SELECT e FROM Employee WHERE e.empId=:eid and e.empName=:ename
List m1(Integer eid,String ename) ;
List m1(String ename, Integer eid) ; //valid
=> While defining method need not follow order for passing param names
   But names must match with named params.

---Named Param Runner--
a. Repository Interface (add below methods)

///--------named params------------------------
    @Query("SELECT e FROM Employee e WHERE e.empName=:ename or
e.empSal>=:esal")
    List<Employee> getEmpsByNamedParam(Double esal,String ename);
    //List<Employee> getEmpsByNamedParam(String ename,Double esal);

    //in operator takes multiple values
    @Query("SELECT e FROM Employee e WHERE e.empId in (:eids)")
    List<Employee> getEmpsBySelectedIds(List<Integer> eids);


b. Runner class
package in.nareshit.raghu.runner;

import in.nareshit.raghu.repo.EmployeeRepo;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import java.util.List;

@Component
public class NamedParamsTestRunner implements CommandLineRunner {
    @Autowired
    private EmployeeRepo repo;
    @Override
    public void run(String... args) throws Exception {
        //repo.getEmpsByNamedParam("A",3.2)
       // repo.getEmpsByNamedParam(3.1,"A")
        repo.getEmpsBySelectedIds(List.of(10,55,36,12))
        .forEach(System.out::println);
    }
}
```

```
-----------------------------------------------------------------
                Association Mapping (HAS-A)/ Multiplicity
                       1...1/1...*/*...1/*...*

*) Two tables are connected using PK-FK
   (Primary Key - Foreign Key Concept)

=> one table PK, another table FK, then relation is created.
=> hint: * side / many side FK Column is created.

=> For *...* (many-to-many) one extra table is created (Join Table)
   with 2 FK columns(JoinColumn, inverserJoinColumn)

  JoinColumn: 1st table PK, child table 1st FK
  inverserJoinColumn: 2nd table PK, child table 2nd FK
```