JDBC , O-R Mapping

Java App ----------> DB s/w (oracle,mysql, postgreSQL and etc..)

What is O-R Mapping or ORM?

Ans) It is the process of mapping java classes with DB table and the properties of java classes with Db table columns and making the objects of java classes representing Db table records having synchronization (i.e if we modify objects ,the db table records will be modified and vice-versa)

Entity class/BO class/PErsistence class

```
public class Customer{
  private int cno;
  private String cname;
  private String cadd;
  private float billAmt;
  //setters & getters
   ...
   ...
}
```

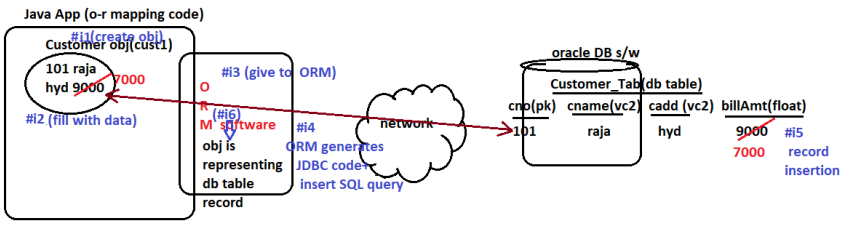
o-r mapping configurations either using xml or annotations

Customer (java class)	Customer_Tab (db table)
cno	cno
cname	cname
cadd	cadd
billAmt	billAmt

java class properties

DB table cols

#i1 to #i6 :: Object based record insertion



=>In O-r Mapping we can perform CURD Opeartions DB table records directly through objects with out dealing with jdbc code and SQL queries.

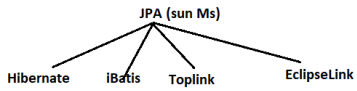
To write o-r mapping persistence logic as objects based DB s/w independent persistence logic with out SQL queries we need the support ORM Frameworks or softwares or tools

Hibernate -----> from SoftTree (Redhat) (Best) (1)
Eclipse link -----> from Eclipse (2)
Toplink ------> from Oracle corp
iBatis ----------> from apache (3)
OJB --------------> from Apache
and etc..

=> What is JPA and how it is related ORM frameworks?

->JPA --->Java Persistence API

Ans) JPA is software specification having set of rules and guidelines to create ORM frameworks or toos or softwares... So switching from One ORM framework to another ORM framework becomes quite easy..



JPA = ORM
(theory) (real implentation)

When should we jdbc and when should we o-r mapping (hibernate) for developing Persistence logic?

Ans) if the App is getting huge amount of data batch by batch for proessing then go for jdbc rather o-r maping becoz to process more records at a time in JDBC we just need one ResultSet object..Where as o-r mapping needs more objects to process more records (at centain project may collapse in the process of creating more objects representing more records)..

eg:: Census , aadhar card issuing and etc..

if the App is dealing with less no.of records then use o-r mapping (hiberante) persistence logic becoz creating less no.of entity objs does not much jvm memory.. more over we can take benifit of other facilities like caching, lazy loading, versioning ,time stamping, DB portability and etc..

eg:: web applicaitons , retail apps , gaming apps

Two types of DB s/ws

1.SQL DB s/ws (eg:: oracle,mysql,postgreSQL and etc..)
2. NO SQL Db s/w (eg:: MongoDB , couchbase and etc..)

=>if the structure of Data is fixed i.e no.of cols are fixed then go for SQL Db s/w

=>Employee db table with 10 colums i.e evry record contains 10 values max
if one employee 25 details to carry then we can not store in model.. if one employee is having only 4 details then the memory for remaining 6 columns in that record will be wasted.

=> SQL Db s/ws are not suitable for storing and managing data with dynamic structure/shema..

=> if the structure of data is not fixed and more over it dynamic then go for NOSQL Db s/ws .. becoz they do not contain db tables having columns and rows.. they maintain collections.. having ability to dynamic Data ... (Unstructred and dynamic schema data)

employee1 with 10 details
employee2 with 20 details
employee3 with 4 details

---

jdbc/spring jdbc , o-r mapping(hibernate) /spring ORM
JAva App ----------> DB s/w (SQL Db s/w)

MongoDB API
JAva App ----------> MongoDB (NoSQL Db s/w)

CouchBase API
JAva App ----------> CouchbaseDB (NoSQL Db s/w)

cassandra api
JAva App ----------> Cassandra DB (NoSQL Db s/w)

note:: Before arrival spring data.. there is no provision in spring to talk with NoSQL Db s/ws..

Spring Data provides unified model or standarad model to interact with SQL Db s/ws in JDBC style or o-r mapping style.. The unified model can also be used to interact with different No SQL Db s/w.. Overall Spring data is one stop solution to develop persistence logic for both SQL and No SQL Db s/w using different approaches.. (It like Universal remote to operate different devices of our home)

spring data JDBC (jdbc style)
Java App ----------> SQL DB s/w

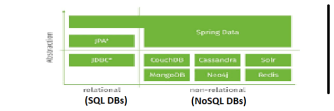
spring data JPA (o-rmapping style)
Java App ----------> SQL DB s/w

spring data MongoDB
Java App ----------> Mongo DB s/w

spring data couchbase
Java App ----------> couch base DB s/w

=>Spring data module is not part of spring framework, it is extension module of spring but can be integrated any spring module Application.

Overview diagram of spring data



Spring data module is one stop solution
to interact with all kinds Db s/w and to develop
all kinds of persistence logics..

we learn ::
spring data jpa (for interacting SQL Db s/w in o-r mapping style)
spring data mongodb (for interacitng mongodb (no SQL Db s/w)

Spring data sub modules

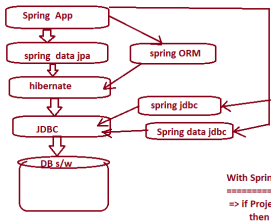
- Spring Data Commons - Core Spring concepts underpinning every Spring Data module.
- Spring Data JDBC - Spring Data repository support for JDBC.
- Spring Data JDBC Ext - Support for database specific extensions to standard JDBC including support for Oracle RAC fast connection failover, AQ JMS support and support for using advanced data types.
- Spring Data JPA - Spring Data repository support for JPA.
- Spring Data KeyValue - Map based repositories and SPIs to easily build a Spring Data module for key-value stores.
- Spring Data LDAP - Spring Data repository support for Spring LDAP.
- Spring Data MongoDB - Spring based, object-document support and repositories for MongoDB.
- Spring Data Redis - Easy configuration and access to Redis from Spring applications.
- Spring Data REST - Exports Spring Data repositories as hypermedia-driven RESTful resources.
- Spring Data for Apache Cassandra - Easy configuration and access to Apache Cassandra or large scale, highly available, data oriented Spring applications.
- Spring Data for Apache Geode - Easy configuration and access to Apache Geode for highly consistent, low latency, data oriented Spring applications.
- Spring Data for Apache Solr - Easy configuration and access to Apache Solr for your search oriented Spring applications.
- Spring Data for Pivotal GemFire - Easy configuration and access to Pivotal GemFire for your highly consistent, low latency/high through-put, data oriented Spring applications.

Community modules

- Spring Data Aerospike - Spring Data module for Aerospike.
- Spring Data ArangoDB - Spring Data module for ArangoDB.
- Spring Data Couchbase - Spring Data module for Couchbase.
- Spring Data Azure Cosmos DB - Spring Data module for Microsoft Azure Cosmos DB.
- Spring Data Cloud Datastore - Spring Data module for Google Datastore.
- Spring Data Cloud Spanner - Spring Data module for Google Spanner.
- Spring Data DynamoDB - Spring Data module for DynamoDB.
- Spring Data Elasticsearch - Spring Data module for Elasticsearch.
- Spring Data Hazelcast - Provides Spring Data repository support for Hazelcast.
- Spring Data Jest - Spring Data module for Elasticsearch based on the Jest REST client.
- Spring Data Neo4j - Spring-based, object-graph support and repositories for Neo4j.
- Spring Data Vault - Vault repositories built on top of Spring Data KeyValue.

Spring Data JPA

=>It is sub module spring data .. that internally uses hibernate to generate objects based o-r mapping persistence logic..



Final Conclusion::

=>As of now Indurstry is prefering to work with spring data jpa to develop o-r mapping persistence logic.. becoz it simplifies work process of hiberante programming.
=> Instead of using plain hibernate directly it is recomanded to use spring data jpa (new trend) or spring ORM (old trend)
=> To develop jdbc style persistence logic use spring jdbc.. instead of working plain jdbc prefer working with spring jdbc.

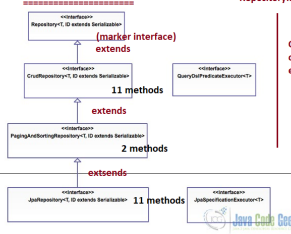
With Spring ORM

=> If Project is having 100 Db tables
then we need to take 100 DAO Interfaces having decl of methods
and 100 DAO Impl classes having implementtion of methods wiht explicitly written persistence logics..

With Spring Data JPA

=>if project is having 100 DB tables
=> Then we need to take 100 Repository Interfaces extending from pre-defined RepositoryInterface

Spring data Repositories



Common Repository Interfaces can be used in both SQL and NOSQL env..

note:: Based on the Repository Interface we supply the Spring DATA JPA Framework internally generates InMemory Proxy class dyamically at rutime having common methods implementation.. inherited pre-defined Repository Interface So we need not to write Persitence logic for those common methods..

Sample Spring Data JPA code::

Employee db table in oracle DB s/w
|--> eno(n) (pk)
|-->ename (vc2)
|-->eadd (vc2)
|-->esalary (float)

=>if db table name and Entity class name are matching then writing @Table is optional
=>if Db table col names and Entity class property names are matching then placing @Column is optional
=> @Entity ,@Id annotations are mandatory annotations..

Entity class

```
@Entity  //mandatory      @Data
@Table(name="Employee")  //optional
public class Employee  implements Serializable{

  @Column(name="ENO")  //optional
  @Id   //mandatory
  private Integer eno;
  @Column(name="ENAME")
  private String ename;
  @Column(name="EADD")
  private String eadd;
  @Column(name="ESALARY")
  private Float  esalary;

  }
```

@Data is lombok api annotation generating setters, getters, toString() , hashCode(),equals() , 0-param consturctor dynamically.

lombok api vedios::
https://www.youtube.com/watch?v=e1_9uPJ31Lw
https://www.youtube.com/watch?v=4hlnLkwgxig

for hibernate videos::

https://www.youtube.com/watch?v=611y-kNr8z0&list=PLVlQHNRLflP8unkTii_FNKDfow3H5C6RL&index=14
(watch upto 14 vedios)

Repository Interface

(I) EmployeeRepo.java (I)

```
public  interface IEmployeeRepo extends  CrudRepository<Employee,Integer>{
}
```

Entity class name ⇩ ⇩ @Id Property Type

Service Interface and service class

EmployeeMgmtService.java (jinterface)

```
public interface  EmployeeMgmtService{
   public   Integer  registerEmployee(EmployeeDTO empDTO);
  }
```

```
@Data
public class EmployeeDTO implements Serializable{
  private Integer eno;
  private String ename;
  private  String eadd;
  private  Float esalary;
  }
```

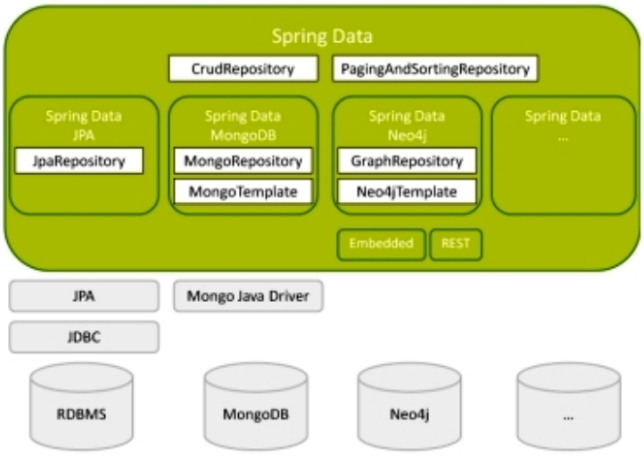
EmployeeMgmtServiceImpl.java (Impl class)

```
@Service("empService")
public class EmployeeMgmtServiceImpl implements  EmployeeMgmtService{

    @Autowired
   private  IEmployeeRepo  empRepo; // Injects  Repository(I) Impl class obj to empRepo property
                                                                (Proxy obj)

   public  Integer registerEmployee(EmployeeDTO dto){

     //convert  EmployeeDTO obj to Employee Entity class obj
       Employee emp=new Employee();
       BeanUtils.copyProperties(dto,emp);
     //use  Repo to save object
       Employee  emp1=empRepo.save(emp);
      return   emp1.getEno();
     }

   }
```

application.properties (spring boot env...)

```
#For dataSoruce cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
# For schema generation/updation
spring.jpa.hibernate.hbm2ddl-auto=update (best)
                                  create
                                  validate
                                  create-drop
```

⇨ (if db table are already avaiable then it uses them by doing required updations otherwise creates new db tables based Entity classes info)



Spring Data JPA First App Development to Save the object

In ORM (o-r mapping)

saving object means :: inserting the record to db table
updating objects means :: updating the record represented by the object
deleting object means :: deleting the record represented by the object
Loading object means :: retrieving record the into the object

step1) create Spring Boot Project adding the following starters (libraries)
=>spring data jpa , lombok , oracle driver

File > new -->spring starter project --->
name :: SpringDataJpaProj1-CRUDRepo
language :: java packing:: jar
version :: 11 tool :: maven
groupId :: nit artifactId :: SpringDataJpaProj1-CRUDRepo
package :: com.nt

step2) add the following jdbc properties and spring data jpa hibernate properties to establish the connection through default HikariCP data source.. and gives instruction to Hibernate configurations.

src/main/resources/application.properties

```
# Datasource cfg
spring.datasource.driver-class-name=oracle.jdbc.driver.OracleDriver
spring.datasource.url=jdbc:oracle:thin:@localhost:1521:xe
spring.datasource.username=system
spring.datasource.password=manager
# hibernate properties
spring.jpa.hibernate.ddl-auto=update
#spring.jpa.show-sql=true
spring.jpa.properties.format_sql=true
spring.jpa.properties.show_sql=true
spring.jpa.properties.dialect=org.hibernate.dialect.Oracle10gDialect
```

(Mandatory)
(optional but recomanded)
(optional)

step3) develop Entity , DTO classes
step4) develope Repository Interface
step4) Develop service interface and service class
step5) develop client app..

Refer SpringDataJPAProj1-CRUDRepo Project in GIT

Methods of CrudRepository(I)

1) <S extends T> S save(S entity)

=>This method performs save obj operation if given id value record is not already present in the DB table .. otherwise it will perform update operation.. Spring Data JPA there is no seperate update(-) method.. we need to use save(-) method itself for update object Operation.

```
@Override
public int registerEmployee(EmployeeDTO dto) {
    //convert dto to Entity
    Employee entity=new Employee();
    BeanUtils.copyProperties(dto, entity);
    //use empRepo
    /*Employee entity1=empRepo.save(entity);
    return entity1.getEno();*/
    return empRepo.save(entity).getEno();
}
```

This method return Entity object representing the record that is inserted/updated.

=>This method internally HB api ses.save(-) method to insert the record or ses.merge(-) method to update the record.

2)<S extends T> Iterable<S> saveAll(Iterable<S> entities)

==>It is batch insertion of records.. To insert "n" records to db table , instead of executing "insert SQL Query for multipletimes by using more network rounds.." we can go for batch insertion where multiple insert SQL queries with different values will be kept in single batch and that batch will be sent to Db s/w at once to reduce network round trips App and Db s/w

usecases:: group ticket reservation , group registrations and etc..

note:: From Java5 onwords Iterable the top most interface for collections earlier it was Collection

Case study :: To reserve 10 tickets executing insert SQL query for 10 times having different values one by one need to use the network b/w java app and Db s/w for 10 times. if we do same operation by using batch insertion.. the network will be used only for 1 time.

In service class

```
@Override
public int[] registerEmployeesGroup(List<EmployeeDTO> listDTO) {
    //convert listDTO to ListEntity obj
    List<Employee> listEntity=new ArrayList();
    listDTO.forEach(dto->{
        Employee entity=new Employee();
        BeanUtils.copyProperties(dto, entity);
        listEntity.add(entity);
    });
    //use empRepo
    List<Employee> listEntity1=(List<Employee>) empRepo.saveAll(listEntity);
    //gathers ids of saved objects
    int ids[]=new int[listEntity1.size()];
    for(int i=0;i<listEntity1.size();++i) {
        ids[i]=listEntity1.get(i).getEno();
    }
    return ids;
}//method
```

In client app

```
int ids[]=service.registerEmployeesGroup(List.of(new EmployeeDTO("rama","hyd",9000.0f),
                                                 new EmployeeDTO("jani","vizag",80000.0f),
                                                 new EmployeeDTO("albert","delhi",70000.0f)
                                                 ));

System.out.println("batch of saved objs ids are ::"+Arrays.toString(ids));
```

---

3) **3) public long count()**

=>returns the count of records that are there in db table.

In service class

```
@Override
public long getEmployeesCount() {
    //use empRepo
    return empRepo.count();
}
```

In Client App

```
System.out.println("Employees count::"+service.getEmployeesCount());
```

4) public boolean existsById(ID id)

=>checks wheather record is available or not based on the given id value.

5) public void deleteById(ID id)

=> deletes the record based on the given id value..

Code in service class

```
@Override
public String removeEmployeeById(int id) {
    //use empRepo
    if(empRepo.existsById(id)) {
        empRepo.deleteById(id);
        return id+" employee is deleted";
    }
    else
        return id+" employee is not deleted";
}
```

Code in client App

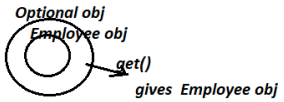
```
System.out.println(service.removeEmployeeById(21));
```

6) ***6) Optional<T> findById(ID id)***

Code in service class

```
@Override
public Optional<EmployeeDTO> getEmployeeById(int id) {
    //use empRepo
    Optional<Employee> opt=empRepo.findById(id);

    Optional optDTO=Optional.empty();
    if(opt.isPresent()) {
        //get Entity obj
        Employee entity=opt.get();
        //covert to DTO
        EmployeeDTO dto=new EmployeeDTO();
        BeanUtils.copyProperties(entity,dto);
        optDTO=Optional.of(dto);
    }
    return optDTO;
}//method
```

=>Optional api is given from java8 .
It is Container obj holding other object.. and useful to check wheather other obj is available or not with support various methods that are supplied. This can be NullChecking with out having the chance of raising NullPointerException.



Code in client app

```
Optional<EmployeeDTO> optDTO=service.getEmployeeById(122);
if(optDTO.isPresent())
    System.out.println("22 employee details "+optDTO.get());
else
    System.out.println("employee not found" );
```

Working with CrudRepository

public void delete(T entity)

Deletes a given entity.
Parameters::
entity - must not be null.
Throws::
IllegalArgumentException - in case the given entity is null.

In service class

```
@Override
    public String removeEmployeeById1(int id) {
        //use  repo;
        Optional<Employee> opt=empRepo.findById(id);
        if(opt.isPresent()) {
            empRepo.delete(opt.get());
            return  "Employee deleted";
        }
        else {
            return "Employee not found to delete";
        }

    }//method
```

In client App

```
System.out.println(service.removeEmployeeById1(125));
```

---

public void deleteAll()

=>Deletes all entities managed by the repository.

public void deleteAll(Iterable<? extends T> entities)

=>Deletes the given entities.
Parameters::
entities - must not be null. Must not contain null elements.
Throws::
IllegalArgumentException - in case the given entities or one of its entities is null.

---

public Iterable<T> findAll()

=>Returns all instances of the type.
Returns::
all entities

code in service class

```
@Override
    public Iterable<EmployeeDTO> getAllEmployees() {
        //use Repo
        Iterable<Employee> itEntities= empRepo.findAll();
        Iterable<EmployeeDTO> itDTOs=new ArrayList();
        itEntities.forEach(entity->{
            EmployeeDTO dto=new EmployeeDTO();
            BeanUtils.copyProperties(entity,dto);
            ((List<EmployeeDTO>) itDTOs).add(dto);
        });
        return itDTOs;
    }
```

In Client App

```
service.getAllEmployees().forEach(System.out::println);
//service.getAllEmployees().forEach(dto-> System.out.println(dto));
```

Exmple on deleteAll(Iterable<T> entities)

Code in service class

```
@Override
public String removeEmployeesByGivenEntities(List<EmployeeDTO> listDTOs) {
    //convert listDTOs to listEntities
    List<Employee> listEntities=new ArrayList();
    listDTOs.forEach(dto->{
        Employee entity=new Employee();
        BeanUtils.copyProperties(dto, entity);
        listEntities.add(entity);
    });
    empRepo.deleteAll(listEntities);
    return  "Multiple Employees are deleted";
}//method
```

Client App

```
try {
        System.out.println(service.removeEmployeesByGivenEntities(List.of(new EmployeeDTO(26),
                                                                           new EmployeeDTO(27),
                                                                           new EmployeeDTO(28))));
    }
    catch(Exception e) {
        System.out.println("Problem in bulk deletion");
        e.printStackTrace();
    }
```

11 am to 12:30noon: HttpUnit

---

public findAllById Iterable<T> findAllById(Iterable<ID> ids)

=>Returns all instances of the type T with the given IDs.If some or all ids are not found, no entities are returned for these IDs.
Note that the order of elements in the result is not guaranteed.
Parameters::
ids - must not be null nor contain any null values.
Returns::
vguaranteed to be not null. The size can be equal or less than the number of given ids.
Throws::
IllegalArgumentException - in case the given ids or one of its items is null.

Code in service class

```
@Override
    public List<EmployeeDTO> getEmployeesByIds(List<Integer> ids) {
        //use empRepo
        List<Employee> listEntities=(List<Employee>) empRepo.findAllById(ids);
        //convert listEntities to listDTOs
        List<EmployeeDTO> listDTOs=new ArrayList();
        listEntities.forEach(entity->{
            EmployeeDTO dto=new EmployeeDTO();
            BeanUtils.copyProperties(entity, dto);
            listDTOs.add(dto);
        });

        return listDTOs;
    }
```

Code in client app

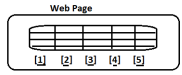
```
service.getEmployeesByIds(List.of(25,26,24)).forEach(System.out::println);
```

Different ways of writing PersInstence logic in Spring Data JPA

a) Using Direct methods of Repository Interfaces like CrudRepository,PagingAndSortingRepository and etc..
b) Using findByXxx() declared in our Repository Interface | select operations (for all ,specfic cols)
c) Using @Query methods of our Repository Interface having HQL/JPQL | Select Operations (All cols)
d) Using @Query methods of our Repository Interface having NativeSQL
e) By Calling PL/SQL procedures/ functions | for any operations
f) Using static and Dynamic Projections (To get with specific col values) | Select operations (specific cols)
g) Using @Query + @Modifying methods of our Repository Interface having HQL/JPQL /NativeSQL to perform Non-select Operations (Custom) | Non-select Operations

PagingAndSortingRepository

=>Sub Interface of CrudRepository and Super Interface of JpaRepository
=>Gives methods to perform Sorting of retrieved records either ASC or in DESC order
=>Gives methods to display records through pagination (displaying multiple records page by page based on the given page No and Pagesize)



methods are::
1. public Page<T> findAll(Pageable pageable)
=> To display records having pagination and /or sorting
2. public Iterable<T> findAll(Sort sort)
=> To display record having Sorting order (ASC /DESC)

1. public Page<T> findAll(Pageable pageable)

Code in service

```
@Override
    public List<EmployeeDTO> getAllEmployees(String property, boolean asc) {
    /*    Sort sort=null;
        if(asc==true) {
            sort=Sort.by(org.springframework.data.domain.Sort.Direction.ASC,"ename");
        }
        else {
            sort=Sort.by(org.springframework.data.domain.Sort.Direction.DESC,"ename");
        }
        List<Employee> listEntities=(List<Employee>) empRepo.findAll(sort); */

        //use  Repo
        List<Employee> listEntities=(List<Employee>) empRepo.findAll(asc?Sort.by(Sort.Direction.ASC, property):
                                                                     Sort.by(Sort.Direction.DESC,property));
        //convert  listEntities to listDTOs
        List<EmployeeDTO> listDTOs=new ArrayList();
        listEntities.forEach(entity->{
            EmployeeDTO dto=new EmployeeDTO();
            BeanUtils.copyProperties(entity, dto);
            listDTOs.add(dto);
        });
        return  listDTOs;
    }
```

Terinay Operator

In Sort class Direction is static inner Enum

In client App

```
//invoke methods
        service.getAllEmployees("eno", false).forEach(System.out::println);
```

Repository(I) (Empty Repository /Marker Interface)
↑ extends
CrudRepository(I) (11)
↑ estends
PagingAndSortingRepository(I) (2)
↑ extends
JpaRepository(I) (11)

Sorting by multiple properties

in servie class

```
@Override
public List<EmployeeDTO> getAllEmployeesBySorting(boolean asc, String... properties) {
    //use Repo
    List<Employee> listEntities=(List<Employee>) empRepo.findAll(asc?Sort.by(Direction.ASC, properties):
                                                                  Sort.by(Direction.DESC,properties));
    //convert listEntities to listDTos
    List<EmployeeDTO> listDTOs=new ArrayList();
    listEntities.forEach(entity->{
        EmployeeDTO dto=new EmployeeDTO();
        BeanUtils.copyProperties(entity, dto);
        listDTOs.add(dto);
    });
    return listDTOs;
}
```

In client app

```
service.getAllEmployeesBySorting(true,"ename","eadd","eSalary").forEach(System.out::println);
```

When we enable Ascending order the priority order is

->special chars (like *,!,+,- and etc..)
-> numbers
->Upppercase alphabets
->Lowercase alphabets

Pagination::

=>It is all about displaying huge no.of records in mulitple pages .page by page.

`public Page<T> findAll(Pageable pageable)`

=>This method takes pageNo,pageSize as inputs in the form Pageable object
and returns Page<T>/Slice <T> obj having List<T>, pageNo, total pages, total records and etc.. info

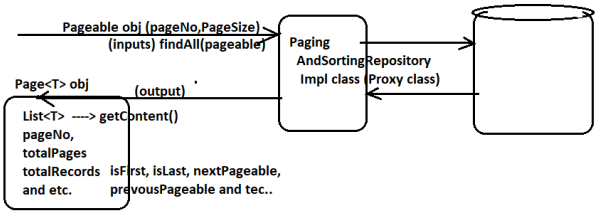
pageNo is 0 based
pageSize divides total no.of records slices
Total records are :: 20 and pagesize 5
then it divides into 4 pages (5,5,5,5)

Slice<T> (I)
↑ extends
Page <T> (I)

To create Pageable object having inputs::

```
Pageable pageable=PageRequest.of(3,5);   //points to 15 to 20(4 th page) records if the total records are 20
```

```
//to get Page obj based on given pageable obj
Page<Employee> page=empRepo.findAll(pageable);
   |-->contains List<T> (15-20 records related Entity objet)
        +
   pageNo, total pages , total records and etc...
```



Example code (we can enable sorting and paging togather)
|-->First it will sort entire db table records and gets the certains page records based on the page number we pass)

In service class

```
@Override
public List<EmployeeDTO> getPageRecords(int pageNo, int pageSize) {
    //Prepared Pageable object having pageNo,pageSize
    Pageable pageable=PageRequest.of(pageNo, pageSize,Direction.ASC,"eno");
    //get Page<T> object
    Page<Employee> page=empRepo.findAll(pageable);
    System.out.println(page.getNumber()+"  "+page.getNumberOfElements()+"  "+page.getTotalPages());
    System.out.println(page.isEmpty()+" "+page.isFirst()+" "+page.isLast());

    //get ListEntities from Page<T> obj
    List<Employee> listEntities=page.getContent();
    //convert listEntities to ListDTOs
    List<EmployeeDTO> listDTO=new ArrayList();
    listEntities.forEach(entity->{
        EmployeeDTO dto=new EmployeeDTO();
        BeanUtils.copyProperties(entity, dto);
        listDTO.add(dto);
    });
    return listDTO;
}
```

In cleint App

```
service.getPageRecords(0,3).forEach(System.out::println);
```

Displaying record multiple records page by page in standalone env..

Code in service class

```
@Override
public void getRecordsByPagination(int pageSize) {
    //get total no.of records
    long count=empRepo.count();
    //get total no.of pages
    int pagesCount=(int) (count/pageSize);

    if(count%pageSize!=0)
        pagesCount++;
    //display records page by page
    for(int i=0;i<pagesCount;++i) {
        Pageable pageable=PageRequest.of(i, pageSize);
        Page<Employee> page=empRepo.findAll(pageable);
        List<Employee> listEntities=page.getContent();
        listEntities.forEach(System.out::println);
        System.out.println(page.getNumber()+1+" of"+page.getTotalPages());
        System.out.println("----------");
    }
```

code in Client app

```
service.getRecordsByPagination(3);
```

JPARepository
==========

=>It sub Interface of PagingAndSortingRepository(I) having 11 methods .. some of the methods Inherited from the CrudRepository(I) are overridden in JPA style providing the same results (Hibernate style)

Repository(I)
↑ extends
CrudRepository(I)
↑ extends
PagingSortingRepository(I)
↑ extends
JpaRepository(I)

Modifier and Type	Method and Description
void	deleteAllInBatch() ✓ Deletes all entities in a batch call.
void	deleteInBatch(Iterable<T> entities) ✓ Deletes the given entities in a batch which means it will create a single Query.
List<T>	findAll()
<S extends T> List<S>	findAll(Example<S> example)
<S extends T> List<S>	findAll(Example<S> example, Sort sort) ✓
List<T>	findAll(Sort sort)
List<T>	findAllById(Iterable<ID> ids)
void	flush() Flushes all pending changes to the database.
T	getOne(ID id) ✓ Returns a reference to the entity with the given identifier.
<S extends T> List<S>	saveAll(Iterable<S> entities)
<S extends T> S	saveAndFlush(S entity) Saves an entity and flushes changes instantly.

=>For example the findAll() method in CrudRepository returns Iterable<T> where as the same of JpaRepository returns List<T>

Iterable<T> findAll() --->CrudRepository(I)
List<T> findAll() --->JpaRepository(I)

<S extends T> Iterable<S> saveAll(Iterable<S> entities) -->CrudRepostory(I)
<S extends T> List<S> saveAll(Iterable<S> entities) -->JpaRepository(I)

=> <S extends T> S saveAndFlush(S entity) is new method in JpaRepository(I)
Saves an entity and flushes changes instantly. Generally The Tx Manager commits the data to Db table while working with CrudRepository, PagingAndSortingReposity methods.. Where as this method directly inserts the record to db table through flush() method call with out any support of Transaction Manager.

=>void flush()
Flushes all pending changes to the database with out taking the support underlying TxManager.

When we add spring data jpa library to spring boot application the HibernateTtransactionManager comes and applies automatically through AutoConfiguration.

=>T getOne(ID id)
Returns a reference to the entity with the given identifier. Depending on how the JPA persistence provider is implemented this is very likely to always return an instance and throw an EntityNotFoundException on first access. Some of them will reject invalid identifiers immediately.
Parameters:
id - must not be null.
Returns:
a reference to the entity with the given identifier.

This method is very much similar to findById(-) method of CrudRepository(I)

note:: while this method @Transactional on the top of service class method or on the top of service class is mandatory..

code ins service class
==========

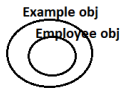
```
@Override
    @Transactional
    public EmployeeDTO fetchEmployeeById(int eno) {
        //use repo
        Employee entity=empRepo.getOne(eno);
        //convert entity to DTO
        EmployeeDTO dto=new EmployeeDTO();
        BeanUtils.copyProperties(entity, dto);
        return dto;
    }
```

In client app

```
//invoke methods
System.out.println("emp details"+service.fetchEmployeeById(21));
```

<S extends T> List<S> findAll(Example<S> example, Sort sort)

=>Example obj is wrapper obj to Entity object .. it will retrieve records from DB table based on given Entity obj's properties values by applying and conditions.. involvies only non null property values of given Example obj based Entity object.



Code in service class
==========

```
@Override
    public List<EmployeeDTO> fetchEmployeesExampleData(EmployeeDTO dto,
                                                       String property , boolean asc) {
        //convert DTO to Entity
        Employee entity=new Employee();
        BeanUtils.copyProperties(dto,entity);
        //prepare Example object
        Example<Employee> ex=Example.of(entity);
        //use repo
        List<Employee> listEntities=empRepo.findAll(ex,
                        asc?Sort.by(Direction.ASC,property):Sort.by(Direction.DESC, property));
    //convert to ListEntities to ListDTO
        List<EmployeeDTO> listDTO=new ArrayList();
        listEntities.forEach(entity1->{
            EmployeeDTO dto1=new EmployeeDTO();
            BeanUtils.copyProperties(entity1, dto1);
            listDTO.add(dto1);
        });
        return listDTO;
    }//method
```

In client app

```
service.fetchEmployeesExampleData(new EmployeeDTO(null,"jani","vizag",80000.0f),"ename", true).forEach(System.out::println);
```

generaed query::

Hibernate: select * from employee where esalary=80000.0 and eadd=? and ename=? order by employee0_.ename asc

eno is not involved in the condition becoz it null

public void deleteInBatch(Iterable<T> entities)

=>Deletes the given entities in a batch which means it will create a single Query.
Parameters: List of entities.

Code in service class

```
@Override
public void removeEmployeesInBatch(List<EmployeeDTO> listDTO) {

    //convert ListDTO to ListEntities
    List<Employee> listEntities=new ArrayList();
    listDTO.forEach(dto->{
          Employee entity=new Employee();
          BeanUtils.copyProperties(dto, entity);
          listEntities.add(entity);
    });
    //use repo
    empRepo.deleteInBatch(listEntities);
}
```

Code in Client App

```
try {
service.removeEmployeesInBatch(List.of(new EmployeeDTO(22),
                    new EmployeeDTO(23),new EmployeeDTO(25)));
          System.out.println("Employees deleted in batch");
    }
    catch(Exception e) {
          System.out.println("Problem in the deletion of Employees");
          e.printStackTrace();
    }
```

Generated SQL query :: Hibernate: delete from employee where eno=? or eno=? or eno=?

---

public void flush()

=> Flushes all pending changes to the database (It is useful to flush the changes (non-select operations) the done in objects to Db table with out taking the TransactionManager).

What is the difference among findAll() methods of CrudRepository , PaginAndSortingRepository and JpaRepositiry?

	Sorting	Paging	return type	Ability to use Example obj
findAll() in CrudRepository	no	no	Iterable<T>	no
findAll() in pagingAndSortingRepository	yes	yes	Iterable<T>	no
findAll() in JpaRepository	yes	no	List<T>	yes

Why there is no update(-) or saveOrUpdate(-) method in Spring Data Repositories?

Ans) Since save(-) is designed to perform both save Object and update object operations.. so seperate methods are not given for update or saveOrUpdate operations.

save(-) method in spring data .. performs save object/record operation , if the object/record is not already available otherwise it performs update object/record operation.

Internal code of save(-) method

```
@Transactional
    @Override
    public <S extends T> S save(S entity) {

        Assert.notNull(entity, "Entity must not be null.");

        if (entityInformation.isNew(entity)) {
            em.persist(entity);     to insert record
            return entity;
        } else {
            return em.merge(entity);    to update record
        }
    }
```

Custom queries or custom logics in spring data jpa

a) Using finder methods in Repository (Select Operations)
b) Using @Query methods in Repository (Select opeations)
c) Using @Query + @Modifying methods in Repository (No-Select Operations)

finder method in Spring data jpa

=>These are abstract methods declared in our Repository Interface , which will be converted into SQL queries dynamically at runtime.

syntax::

```
public  <RT>  find/By<PropertiesName(s)><Condition> (params ...)
```

=>Supports only Select operaitons
=>We can write multiple conditions based finder methods
=>Two types select operations using finder methods
a) Entity operations (Getting all col values with different conditions) of db table
b) Scalar Operations (Getting specific col values or aggregate operations with diffeent conditions) of db table
(These operatios are also called Projections)

=> These are the custom methods declared in our repository.. and the code will be generated based on name of the method having SQL query in dynamic proxy class..

=>if we declare finder method with out condition ,It will generated is or = condition in the given property/column.

=> we can ctx.getBean(<bean class name/type>) to get Spring bean object when we can not pass bean id ..this is very useful to get Proxy class obj from IOC container when that Proxy class generation and object is happening dynamically at runtime as InMemory activities.

eg:: IEmployeeRepo repo=ctx.getBean(IEmployeeRepo.class);
Repository Interface name becomes spring bean having its class name as the default bean id at runtime.. we can not use that class name or bean id in ctx.getBean(-) becoz that is generated dynamically at runtime, So we use ctx.getBean(< class/super type/name>) to get such spring bean class obj.

Examples

Code in Repository(I)

```
public interface IEmployeeRepo extends JpaRepository<Employee, Integer> {
    //SELECT ENO,ENAME,EADD,ESALARY FROM EMPLOYEE WHERE EADD=?
    public List<Employee> findByEadd(String eadd);
}
```

code in Client app

```
//invoke method
repo.findByEadd("hyd").forEach(System.out::println);
```

if we pass wrong property name in finder methods .. the we get
org.springframework.beans.factory.BeanCreationException: Error creating bean with name 'IEmployeeRepo' defined in com.nt.repository.IEmployeeRepo defined in @EnableJpaRepositories declared on JpaRepositoriesRegistrar.EnableJpaRepositoriesConfiguration: Invocation of init method failed; nested exception is java.lang.IllegalArgumentException: Failed to create query for method public abstract java.util.List com.nt.repository.IEmployeeRepo.findByEadd1(java.lang.String)! No property eadd1 found for type Employee! Did you mean 'eadd'?

note:: the property name in finder Method and param name need not to match by default. if u r getting error then we need to change settings in eclipse..

right click on project ---> properties ---> java compiler --> ☑ Store information about method parameters (usable via reflection)

```
public List<Employee> findByEadd(String addrs);
```

property name — param name — note:: findBy prefix in method name is mandatory otherwise exception will come

---

```
//SELECT ENO,ENAME,EADD,ESALARY FROM EMPLOYEE WHERE ENAME=?
public List<Employee> findByEname(String ename);
```

```
repo.findByEname("raja").forEach(System.out::println);
```

Keyword	Sample	JPQL snippet
And	findByLastnameAndFirstname	… where x.lastname = ?1 and x.firstname = ?2
Or	findByLastnameOrFirstname	… where x.lastname = ?1 or x.firstname = ?2
Is,Equals	findByFirstname,findByFirstnameIs,findByFirstnameEquals	… where x.firstname = 1?
Between	findByStartDateBetween	… where x.startDate between 1? and ?2
LessThan	findByAgeLessThan	… where x.age < ?1
LessThanEqual	findByAgeLessThanEqual	… where x.age <= ?1
GreaterThan	findByAgeGreaterThan	… where x.age > ?1
GreaterThanEqual	findByAgeGreaterThanEqual	… where x.age >= ?1
After	findByStartDateAfter	… where x.startDate > ?1
Before	findByStartDateBefore	… where x.startDate < ?1
IsNull	findByAgeIsNull	… where x.age is null
IsNotNull,NotNull	findByAge(Is)NotNull	… where x.age not null
Like	findByFirstnameLike	… where x.firstname like ?1
NotLike	findByFirstnameNotLike	… where x.firstname not like ?1
StartingWith	findByFirstnameStartingWith	… where x.firstname like ?1 (parameter bound with appended %)
EndingWith	findByFirstnameEndingWith	… where x.firstname like ?1 (parameter bound with prepended %)
Containing	findByFirstnameContaining	… where x.firstname like ?1 (parameter bound wrapped in %)
OrderBy	findByAgeOrderByLastnameDesc	… where x.age = ?1 order by x.lastname desc
Not	findByLastnameNot	… where x.lastname <> ?1
In	findByAgeIn(Collection<Age> ages)	… where x.age in ?1
NotIn	findByAgeNotIn(Collection<Age> age)	… where x.age not in ?1
True	findByActiveTrue()	… where x.active = true
False	findByActiveFalse()	… where x.active = false
IgnoreCase	findByFirstnameIgnoreCase	… where UPPER(x.firstame) = UPPER(?1)

Code in Repository
==========

```
public interface IEmployeeRepo extends JpaRepository<Employee, Integer> {
        //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE  EADD=?
    public  List<Employee>  findByEadd(String addrs);
    //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE  ENAME=?
    public  List<Employee>  findByEname(String name);
    public  List<Employee>  findByEnameIs(String name);
    public  List<Employee>  findByEnameEquals(String name);

  //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE  ESALARY<?
    public  List<Employee>  findByEsalaryLessThan(float startSalary);

  //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE  ESALARY>=?
    public  List<Employee>  findByEsalaryGreaterThanEqual(float startSalary);

    //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE  ENAME LIKE 'r%'
    public  List<Employee>  findByEnameLike(String chars);

  //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE  ENAME LIKE 'r%'
    public  List<Employee>  findByEnameStartingWith(String initialChars);

  //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE  ENAME LIKE '%a%'
    public  List<Employee> findByEnameContaining(String chars);

  //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE EADD IN('hyd','mumbai','delhi')
    public  Iterable<Employee>  findByEaddIn(Collection<String> cities );

    //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE EADD NOT IN('hyd','mumbai')
    public  Iterable<Employee>  findByEaddNotIn(Collection<String> cities );

    //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE EADD IS NULL
    public  Iterable<Employee>  findByEaddIsNull();

  //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE EADD ISNOT NULL
    //public  Iterable<Employee>  findByEaddIsNotNull();
    public  Iterable<Employee>  findByEaddNotNull();

  //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE UPPER(EADD)=UPPER('hyd')
    public  Iterable<Employee>  findByEaddIgnoreCase(String city);

    //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE EADD='hyd' ORDER BY ENO ASC
    //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE EADD='hyd' ORDER BY ENO
    public  Iterable<Employee>   findByEaddOrderByEnoAsc(String city);
    public  Iterable<Employee>   findByEaddOrderByEno(String city);

  //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE ENAME='raja' AND EADD='hyd'
    public   List<Employee>  findByEnameAndEadd(String name,String addrs);

    //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE ESALARY=10000 OR EADD='hyd'
    public   List<Employee>  findByEsalaryOrEadd(float salary,String addrs);

    //SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE ESALARY BETWEEN 10000 AND 20000
    public   List<Employee>  findByEsalaryBetween(float startSalary,float endSalary);




}
```

Code in Client app

```
@SpringBootApplication
public class SpringDataJpaFinderMehodsEntityOperationsApplication {

    public static void main(String[] args) {
        //get IOC container
        ApplicationContext ctx=SpringApplication.run(SpringDataJpaFinderMehodsEntityOperationsApplication.class, args);
        //get Proxy class object
        IEmployeeRepo  repo=ctx.getBean(IEmployeeRepo.class);
        System.out.println("proxy class name::"+repo.getClass());
        //invoke method
        // repo.findByEadd("hyd").forEach(System.out::println);
         //repo.findByEname("raja").forEach(System.out::println);
          //repo.findByEnameIs("raja").forEach(System.out::println);
        //repo.findByEnameEquals("raja").forEach(System.out::println);
         //repo.findByEsalaryLessThan(10000).forEach(System.out::println);
         //repo.findByEsalaryGreaterThanEqual(10000).forEach(System.out::println);
        //repo.findByEnameLike("r%").forEach(System.out::println);
        //repo.findByEnameStartingWith("r").forEach(System.out::println);
        //repo.findByEnameContaining("a").forEach(System.out::println);
        //repo.findByEaddIn(List.of("hyd","mumbai","delhi")).forEach(System.out::println);
         //repo.findByEaddNotIn(List.of("hyd","mumbai")).forEach(System.out::println);
         //repo.findByEaddIsNull().forEach(System.out::println);
        //repo.findByEaddIsNotNull().forEach(System.out::println);
        //repo.findByEaddNotNull().forEach(System.out::println);
         //repo.findByEaddIgnoreCase("hyd").forEach(System.out::println);
        //repo.findByEaddOrderByEno("hyd").forEach(System.out::println);
        //repo.findByEnameAndEadd("raja", "hyd").forEach(System.out::println);
        //repo.findByEsalaryOrEadd(10000, "hyd").forEach(System.out::println);
         repo.findByEsalaryBetween(10000, 100000).forEach(System.out::println);
    }

}
```

spring Data JPA finder methods using multiple properties

Code in Repository Interface

```
//SELECT ENO,ENAME,EADD,ESALARY FROM EMPLOYEE WHERE ENAME='raja' AND EADD='hyd'
    public List<Employee> findByEnameAndEadd(String name,String addrs);

    //SELECT ENO,ENAME,EADD,ESALARY FROM EMPLOYEE WHERE ESALARY=10000 OR EADD='hyd'
    public List<Employee> findByEsalaryOrEadd(float salary,String addrs);

 //SELECT ENO,ENAME,EADD,ESALARY FROM EMPLOYEE WHERE (ENO>=100 AND ENAME LIKE '%j') OR (ESALARY BETWEEN 10000 AND 20000)
 public List<Employee> findByEnoGreaterThanEqualAndEnameEndingWithOrEsalaryBetween(int startEno,String namelastChars,float
startSalary,float endSalary);


//SELECT ENO,ENAME,EADD,ESALARY FROM EMPLOYEE WHERE ENAME IN('raja','rani','suresh') OR EADD IN('hyd','mumbai','delhi')
 public List<Employee> findByEnameInOrEaddIn(Collection<String> names,Collection<String> addresses);
```

Code in Client app

```
//repo.findByEnameStartingWithAndEaddStartingWith("r","h").forEach(System.out::println);
//repo.findByEnameStartingWithAndEaddStartingWith("r","h").forEach(System.out::println);
//repo.findByEnoGreaterThanEqualAndEnameEndingWithOrEsalaryBetween(100, "a", 10000, 20000).forEach(System.out::println);
repo.findByEnameInOrEaddIn(List.of("raja","rani","suresh"),List.of("hyd","vizag","delhi")).forEach(System.out::println);
```

note:: Writing finder methods with multiple properties and conditions is bit complex and also makes the method names very long , so prefer using @Query method as alternate.

Scalar Operations (Selecting specific single or multiple col values) Using Spring Data JPA Projections

=>In SQL Writing Queries by speclifying specific col names is called Projections.

Spring Data JPA Supports two types of Projections

a) Static Projections
(Always give fixed specific col values)

b) Dynamic Projections
(Can give varying specific col values)

Procedure to work static Projections

a) Define TypeView Interface in seperate file having method declaration on specifc getter Properties.

ResultView.java

```
package com.nt.repository;
//TypeView Interface for static Projections
interface ResultView{
    public Integer getEno();
    public String getEadd();
}
```

b) Declare the finder method inside the Repository Interface having the TypeView Interface as the return type.

```
public interface IEmployeeRepo extends JpaRepository<Employee, Integer> {
    //SELECT ENO,EADD FROM EMPLOYEE WHERE EADD IN('hyd','mumbai','delhi')
    public List<ResultView> findByEaddIn(Collection<String> cities);

}
```

c) Invoke the finder method from the Client app

```
//invoke methods
/*List<ResultView> list=repo.findByEaddIn(List.of("hyd","vizag","mumbai"));
list.forEach(view->{
    System.out.println(view.getEno()+"  "+view.getEadd());
});*/
    (or)
repo.findByEaddIn(List.of("hyd","vizag","mumbai")).forEach(view->System.out.println(view.getEno()+"  "+view.getEadd()));
```

note:: Here Two InMemory Proxy classes will be geneated by SpringData JPA
a)Proxy class implementing our Reposotry Interface (eg: IEmployeeRepo)
b) Proxy class implementing the TypeView(eg: ResultView) interface for static Projections

Assignment:: get Eno,ename, salary properties(col values) based given initial characters of employee name

Core java Recap
==========

```
class Person {
  ...
}

class Student extends Person{
  ..
}

class Employee extends Person{
  ...
}

class Customer extends Person{
  ...
}
```

option1:: public Object display(Object object){
...
}
good --> but we can also pass other than Person class hierarchy class objs also as the argument values

option2:: public Person display(Person person){
...
}
good --> Here we can pass only Person or its sub class objs as the argument value..

=>In both options, if we want to return Passed object.. then we need to go for typecasting while calling the method.. So code is not type safe ..(i.e there is a possibility of getting ClassCastException)

eg1:Customer cust=(Customer)display(new Customer());
eg2:Studnet stud=(Student)display(new Student());

option3 :: public <T> display(Class <T> clazz){
...
..
}
Good --> Code is type safe becoz of generics..
(Here we can pass any object other than Person class Hierarchy..)

Code is type safe .. So no ClassCastingException will be raised..

```
Customer customer=display(Customer.class);
Date date=display(Date.class);
Employee emp=display(Employee.class);
```

option4 :: public <T extends Person> <T> display(Class <T > clazz){
....
}
More Good --> Code is type safe becoz of generics.. we can pass and get only Person Hierarchy class objs

```
Customer customer=display(Customer.class);
Date date=display(Date.class);
Employee emp=display(Employee.class);
```

Spring Data JPA Finder methods with Dynamic Projections (Getting Varying specific multipe col values)
==========

step1) Take TypeView Interfaces as show below having inheritence relation ship

View.java
```
inerface View{

}
```

ResultView1.java
```
interface ResultView1 extends View{
   public String getEname();
   public String getEadd();
}
```

ResultView2.java
```
interface ResultView2 extends View{
   public Integer getEno();
   public Float getEsalary();
}
```

ResultView2.java
```
interface ResultView3 extends View{
    public Integer getEno();
    public String getEname();
    public Float getEsalary();
}
```

step2) define finder method in the Repository Interface having List<T extends View> as the return type

EmployeeRepo.java
```
interface EmployeeRepo extends JpaRepository<Employee.class,Integer>{
   public <T extends View> List<T> findByEadd(String addrs, Class<T> clazz );
}
```

step3) Invoke the finder method in the client app..

```
List<ResultView1> list1=repo.findByEadd("hyd",ResultView1.class);
    list1.forEach(view1->{ s.o.p(view1.getEname()+"  "+view1.getEadd()); })
  (or)
List<ResultView2> list2=repo.findByEadd("hyd",ResultView2.class);
    list1.forEach(view2->{ s.o.p(view2.getEno()+"  "+view2.getEslary()); })
  (or)
List<ResultView3> list3=repo.findByEadd("hyd",ResultView3.class);
    list3.forEach(view3->{ s.o.p(view3.getEno()+"  "+view3.getEname()+"  "+view3.getEsalary()); })
```

Limitations of Spring Data Jpa finder methods
==========

a) supports only select operations i.e no support for non-select operations.
b) Not suitable for applying more than one property based condition. if we do so .. the length of method names will be increased.
c) We must be method names by following some convention (findBy<property><condition>) , otherwise no use.
d) Aggregate operations are not possible.
e) Working Scalar Operations (Projections) is bit complex
d) Code is not so redable...

note:: To overcome all these problems .. take the support of @Query methods..

conclusion:: use Spring data jpa finder methods for single property and single condition based select Operations.

@Query methods
==========

=>we can write either HQL/JPQL or NativeSQL queries directly on the top of the method declared in Repository Interface
=> method name can be taken as needed (No Naming Convention to fallow)

Native SQL = underlying DB s/w Specific SQL

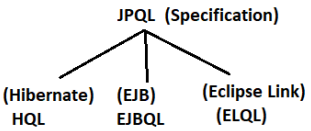
```
@Query(" HQL /JPQL or Native SQL query")
public <return type> <method name>(parmas ....)
```

=>Supports both select and non-select operations (expect insert operation)
=> We can select 0 or more rows with our choice conditions with our having lengthy method names..
=> HQL /JPQL supports both named(:<name>) and positional parameters(?1,?2,?3..)
=> Supports Aggregate operations ..
=> Allows to call PL/SQL procedures and functions..
=> supports to work with Joins to get Two db tables data...
and etc...

Why the @Query method does not support insert operation?

Ans) while working HQL /JPQL or Native SQL Insert Queries we can not involve the HB Generators to generate the Identity values.. They will be involved only while working with repo.save(-) or repo.saveAll(-) or repo.saveAndFlush() method.
conclusion:: use repo.save(-) and repo.saveXxx(-) methods to perform save object operation (To insert record)

HQL/JPQL
=========
HQL ->Hibernate query Language
JP QL --> Java Persistence Query Language



HQL/JPQL
=========
-> It is object based Query language
-> These queries are written by specifying Entity class name and its properties
-> supports both select and non-select operations (expect insert operation)
-> supports both Entity and scalar select operations
-> These DB s/w independent Queries
-> Supports all condition clauses
-> Supports both positional(?1,?2,?3,...) and named Params (:<name1>,:<name2>,...)
-> Every HQL query will be converted into underlying DB s/w specific SQL query internally
-> HQL keywords are not case-sensitive , but the class names and variable names specified in the HQL are case-sensitive

Db table name
SQL> SELECT * FROM EMPLOYEE
HQL/JPQL> FROM com.nt.entity.Employee (or)
> FROM Employee e (or)
> FROM Employee as e (or)
> SELECT e FROM Employee as e
Entity class name — alias name

=>In HQL /JPQL Queries ,placing SELECT Keyword is optional when we are slecting all colum values from Db table.. otherwise it is mandatory to place.

SQL> SELECT ENO,ENAME FROM EMPLOYEE WHERE ENO>=? AND ENO<=?
HQL/JPQL> SELECT eno,ename FROM Employee WHERE eno>=?1 AND eno<=?2
HQL/JPQL> SELECT e.eno,e.ename FROM Employee as e WHERE e.eno>=?1 AND e.eno<=?2

?,?,.. are called Positional params
?1,?2,.. are called ordinal Positional params

note:: From Hibernate 5.2 onwords there is no support jdbc style plain positional params (?)
we should always use JPA style ordinal Positional params (?1)

db table name — col name — ordinal positional param
SQL> DELETE FROM EMPLOYEE WHERE EADD=?
HQL/JPQL> DELETE FROM Employee WEHRE eadd=?1
HQL/JPQL> DELETE FROM Employee WEHRE eadd=:addrs
Entity class name — named param

=>The @Query methods of Repository(I) will have flexible signatures i.e we can take our choice method names, retur types and paramer names/types.

Example
========

Code in Reposiory Interface::

```
//@Query("FROM com.nt.entity.Employee ")
//@Query("FROM Employee ")
//@Query("FROM Employee e ")
@Query("SELECT e FROM Employee e ")
public List<Employee> fetchAllEmployees();
```

Code in client app

```
repo.fetchAllEmployees().forEach(System.out::println);
```

```
@Query(" FROM Employee WHERE eadd=?1")
//@Query(" FROM Employee WHERE eadd=:city")
public List<Employee> fetchAllEmployeesByAddrs(String addrs);
```

```
repo.fetchAllEmployeesByAddrs("hyd").forEach(System.out::println);
```

note:: while working with name params .. if the named param is matching with method param then there is no need of placing @Param annotaion (make sure that Store information about method parameters (usable via reflection) is slected) (Right click on the Project -->properties --> java compiler --> ...)

otherwise we must place @param in method param as shown below. (if not placed then IllegalStateException will come)

```
@Query(" FROM Employee WHERE eadd=:city")
public List<Employee> fetchAllEmployeesByAddrs(@Param("city") String addrs);
```

Code in Repository

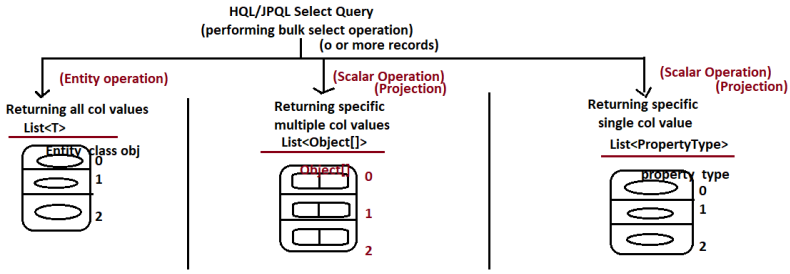
```
@Query("FROM Employee WHERE esalary>=:start and esalary<=:end")
public List<Employee> fetchAllEmployeeBySalaryRange(float start, float end);
```

Code in Client App

```
repo.fetchAllEmployeeBySalaryRange(10000, 20000).forEach(System.out::println);
```

note:: In HQL/JPQL query
a) we can not place jdbc style plain positional params(?)
b) we can not place both named and positional params at a time
c) we can take parameters only representing input values and but not reprsensting HQL key words , Entity class names and property names

=>FROM Employee WHERE esalary between ?1 and ?2 //valid
=>FROM Employee WHERE esalary between :min and :max //valid
=>FROM Employee WHERE esalary between ?1 and :max //invalid (we can not place both ordinal Postional and named params at a time)
=> FROM Employee WHERE 1? beween ?2 and ?3 // invalid
=> FROM Employee WHERE ?1 ?2 ?3 and ?4 // invalid
=> FROM :p1 WHERE :p2 :p3 :p4 and :p5 // invalid



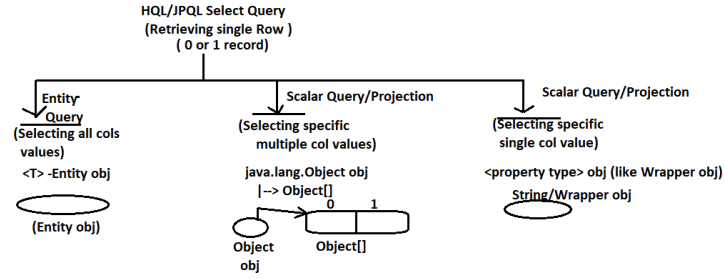
Examples
=========

Code in Repository Interface

```
//---------- Retrieving specific multiple col values (scalar /Projections in BulkSelect ) ----------
@Query("SELECT eno,ename,esalary FROM  Employee WHERE eadd in(:city1,:city2,:city3)")
public List<Object[]>  fetchEmpDetailsByCities(String city1,String city2,String city3);

@Query("SELECT eno,ename FROM  Employee WHERE ename like :initialLetters%")
public List<Object[]> fetchEmpDetailsByNameInitialLetters(String initialLetters);

//---------- Retrieving specific single col values (scalar /Projections in BulkSelect ) ----------
@Query("SELECT ename FROM  Employee WHERE eno>=:start and eno<=:end")
public  List<String>  findEmpNamesByEnoRange(int start ,int end);

@Query("SELECT esalary FROM  Employee WHERE ename IN(:name1,:name2,:name3)")
public  List<Float>  findEmpSalaryByEnames(String name1,String name2,String name3);
```

Code in client app

```
//----------Retrieving specific multiple col vlaues (scalar queries/projections in BulkSelect)----------
        repo.fetchEmpDetailsByCities("hyd", "delhi","mumbai").forEach(row->{
              System.out.println(row[0]+"  "+row[1]+"  "+row[2]);
        });

        repo.fetchEmpDetailsByNameInitialLetters("r").forEach(row->{
              System.out.println(row[0]+"  "+row[1]);
        });

        //----------Retrieving specific single col vlaues (scalar queries/projections in BlukSelect)----------
         repo.findEmpNamesByEnoRange(10,100).forEach(System.out::println);
         repo.findEmpSalaryByEnames("raja", "rani","jani").forEach(System.out::println);
```



When we have findById(-) or getOne(-) methods directly in pre-defined Repository Interfaces then why should we go for @Query methods to deal with Entity Select Query that performs single row operation?

Ans) It is useful to retrive single row/record based on other unique col values as the condition values like based on mail id , passport number and etc..

Example Code
==========

Code in Repository::

```
//---------- Retrieving all col values of single row (Entity Operation in SingleRow Select query ) ----------
        @Query(" FROM  Employee WHERE eno=:no")
        public Employee  findEmpAllDetailsByEno(int no);

            //---------- Retrieving specific multiple   col values of singlerow (scalar/Projection Operation in SingleRow Select query ) ----------
        @Query("SELECT eno,eadd FROM  Employee WHERE eno=:no")
        public    Object  findEmpPartialDetailsByEno(int no);

        //---------- Retrieving specific single   col value of singlerow (sclar/Projection Operation in SingleRow Select query ) ----------
        @Query("SELECT esalary FROM  Employee WHERE eno=:no")
        public    float  findEmpSalaryByEno(int no);
```

Code in client app

```
//---------- Retrieving all col values of single row (Entity Operation in SingleRow Select query ) ----------
        Employee emp=repo.findEmpAllDetailsByEno(29);
        System.out.println(emp);

         //---------- Retrieving specific multiple   col values of singlerow (scalar/Projection Operation in SingleRow Select query ) ----------
        Object obj=repo.findEmpPartialDetailsByEno(29);
        Object row[]={Object[])obj;
        System.out.println("details are::"+row[0]+"  "+row[1]);

       //---------- Retrieving specific single   col value of singlerow (scalar/Projection Operation in SingleRow Select query ) ----------
        float salary=repo.findEmpSalaryByEno(29);
        System.out.println("salary is ::"+salary);
```

=>HQL/JPQL Supports Aggregate functions

Code in Repository

```
//------------------- Supports aggregate functions -------------------
        @Query("SELECT COUNT(*) FROM Employee")
        public int getEmpsCount();

        @Query("SELECT COUNT(*),MAX(esalary),MIN(esalary),avg(esalary) FROM Employee")
        public Object getEmpsAggerateData();
```

Code in client App

```
// ------------------- HQL/JPQL with aggregate results ---------------------
System.out.println("Employees count ::"+repo.getEmpsCount());
Object results[]=(Object[]) repo.getEmpsAggerateData();
System.out.println("Emps count::"+results[0]+" MAX salary:: "+results[1]+" Min salary ::"+results[2]+"  "+"Avg salary::"+results[3]);
```

Non-Select Operations Using @Query methods

=>we must use @Query ,@Modifying annotations

=>if u r working @Service class then there is no need of taking @Transactional otherwise we must add @Transactional in the Repository (I) on the top of method

=>Supports update,delete non-select operations using HQL query.. but there is no insert HQL query

So use repo.save(-) for insert operation.

why repo.save(-) for insert operation?

Ans) =>Allows to work with generators to generate the Id property or PK column value dynamically. Using HQL insert ,we can not id values dynamically using generators. So HQL insert is bad.

Examples

Code in Repository

```
//-------------- HQL/JPQL Non-select operations (except insert operation) ----------------------
        @Query("UPDATE Employee SET esalary=esalary+ :amount WHERE esalary<=:startSalary")
        @Modifying
        @Transactional
            public int updateEmpSalaryByAmount(float startSalary, float amount);

        @Query("DELETE FROM Employee WHERE eadd IS NULL")
        @Modifying
        @Transactional
        public int deleteEmpsHavingNoAddrs();
```

if Non-select persistence operations are not taking place in Transactional env.. then ***we get exception***

javax.persistence.TransactionRequiredException: Executing an update/delete query

Code in Client App

```
//-------------------- HQL/JPQL Non-Select Operations ----------------------
            int count=repo.updateEmpSalaryByAmount(10000,1000);
            System.out.println("no.of records that are effected::"+count);

            int count1=repo.deleteEmpsHavingNoAddrs();
            System.out.println("no.of emps that are deleted::"+count1);
```

@Query methods having Native SQL query

=> if certain operations are not possible with HQL/JPQL Queries then we can develop them through Native SQL Queries (DB S/w specific SQL Queries)

=> NativeSQL queries based Persistence logic is DB s/w dependent persistence logic

=> NativeSQL Queries based Operations are

->insert operation

-> DB s/w specific aggregate operations

-> date and time operations like sysdate(oracle) , now()(mysql) and etc..

-> calling PL/SQL procedures and functions..

Code in Repository

```
@Query(value="SELECT SYSDATE FROM DUAL",nativeQuery=true)
        public String getSysDate();

        @Query(value="INSERT INTO EMPLOYEE VALUES(?,?,?,?)",nativeQuery=true)
        @Modifying
        @Transactional
        public int insertEmployee(int no,float salary, String addrs ,String name);
```

Code in client app

```
//------------------------ Native SQL query --------------------
            System.out.println("sys date::"+repo.getSysDate());
            int count=repo.insertEmployee(678, 9999.0f, "hyd","mohit");
            if(count==0)
                System.out.println("record not inserted");
            else
                System.out.println("record inserted");
```

While working Native SQL queries we can take 3 types of params in the SQL query

a) Positional params (?,?,?)

b) Ordinal Positional params (?1,?2,?3,...)

c) Named Params (:<name1>,:<name2>,....)

Calling PL/SQL Procedures/ Functions

==========

multiple modules..

usecase1:: Instead of writing authentiation logic in every module .. write it only for 1 time in Db s/w as PL/SQL procedure /function and use it/call it in multiple modules.

usecase2:: Instead of writing attendence calculation logic in every module .. write it only for 1 time in Db s/w as PL/SQL procedure /function and use it/call it in multiple modules.

note:: PL/SQL procedure does not return value. where as PL/SQL function returns a value.

note:: PL/SQL procedure or function contains parmaeters with 3 modes

a) IN mode (default)

b) OUT mode

c) INOUT mode

PL/SQL Logic ::

---

y:=x*x; x ->IN mode parameter , y -->OUT mode parameter

PL/SQL Logic ::

---

x:=x*x; x ->INOUT mode parameter

=>if PL/SQL procedure wants to return n outputs then we need to take 10 out params.

=>if PL/SQL function wants to return 10 outputs then we need to take 9 out params and 1 return value.

In Typical java Project

=> 60 to 70% Persistence logic is written using SQL/HQL queries + o-rmapping logic

=> 30% to 40% Persistence logic is written using PL/SQL procedure or function

=>In Spring data we can call PL/SQL procedure /function with the support EntityManager, which comes in every Spring data JPA Application through AutoConfiguration.

=>PL/SQL programming is specific to each DB s/w.. So its persistence logic is Db s/w dependent Persistence logic always.

Calling PL/SQL procedure of Oracle from Spring data JPA Application

==========

step1) create PL/SQL procedure in oracle..

Open SQL Developer --> right click on Procedures ---> new

name :: P_GET_EMPS_BY_SALRANGE use + symbol for adding parameters.

STARTSALARY IN NUMBER

ENDSALARY IN NUMBER

DETAILS OUT SYS_REFCURSOR--> compile

```
CREATE OR REPLACE PROCEDURE P_GET_EMPS_BY_SALRANGE
(
 STARTSALARY IN NUMBER
, ENDSALARY IN NUMBER
, DETAILS OUT SYS_REFCURSOR
) AS
BEGIN
 OPEN DETAILS FOR
   SELECT ENO,ENAME,EADD,ESALARY FROM EMPLOYEE WHERE ESALARY>=STARTSALARY AND ESALARY<=ENDSALARY;

END ;
```

=>In Oracle's PL/SQL Programming to hold bunch of records given by SELECT SQL query , we need to take the support of Cursor type OUT parameter.

=>SYS_REFCUSOR is Cursor Data type in Oracle whose variable/out parameter can hold bunch of records given by select SQL query

eg:

```
details SYS_REFCUSOR;
  open details for
    SELECT * FROM EMPLOYEE WHERE <condition>
```

Here the records given SELECT SQL query will be stored in "details" Cursor variable.

step2) Inject or get EntityManager object either in the service class /client app and use that object for calling PL/SQL procedure..

In service class

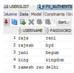
==========

//EmployeeMgmtServiceImpl.java

```
package com.nt.service;

import java.util.ArrayList;
import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.ParameterMode;
import javax.persistence.StoredProcedureQuery;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.nt.dto.EmployeeDTO;
import com.nt.entity.Employee;

@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
	@Autowired
	private EntityManager em;


	/*CREATE OR REPLACE PROCEDURE P_GET_EMPS_BY_SALRANGE
	(
	  STARTSALARY IN NUMBER
	, ENDSALARY IN NUMBER
	, DETAILS OUT SYS_REFCURSOR
	) AS
	BEGIN

	  OPEN DETAILS FOR
	    SELECT ENO,ENAME,EADD,ESALARY FROM EMPLOYEE WHERE ESALARY>=STARTSALARY AND ESALARY<=ENDSALARY;

	END ; */

	@Override
	public List<EmployeeDTO> fetchEmpsBySalaryRange(float startSalary, float endSalary) {
		//create StoredProcedure Object
		StoredProcedureQuery procedure=em.createStoredProcedureQuery("P_GET_EMPS_BY_SALRANGE",Employee.class);
		//register PL/SQL Procedure Parameters
		procedure.registerStoredProcedureParameter(1, Float.class, ParameterMode.IN);
		procedure.registerStoredProcedureParameter(2, Float.class, ParameterMode.IN);
		procedure.registerStoredProcedureParameter(3, Employee.class, ParameterMode.REF_CURSOR);
		//set values to IN params
		procedure.setParameter(1, startSalary);  //startSalary
		procedure.setParameter(2,endSalary);  //endSalary
		//call PL/SQL procedure
		List<Employee> listEntities=procedure.getResultList();
		//convert listEntities to listDTO
		List<EmployeeDTO>  listDTO=new ArrayList();
		listEntities.forEach(entity->{
			EmployeeDTO dto=new EmployeeDTO();
			BeanUtils.copyProperties(entity,dto);
			listDTO.add(dto);
		});
		return listDTO;
	}//method
}//class
```

=>Spring Data JPA does not give direct provision to call PL/SQL function of DB s/w.. So we need think about using plain JDBC code (CallableStatement obj) by unwrapping Session object from EntityManager.

note:: PL/SQL procedure does not return a value.. where as PL/SQL function returns a value
note:: if u want to get 10 results from PL/SQL procedure then we need to use 10 no.of out params
Similarly if u wan to get 10 results from PL/SQL function then we need to use 9 no.of out params and 1 return value.



PL/SQL Function in oracle

```
-----------------------------------------
CREATE OR REPLACE FUNCTION FX_AUTHENTICATION( UNAME IN VARCHAR2 , PWD IN VARCHAR2) RETURN VARCHAR2 AS
  CNT NUMBER(5);
  RESULT VARCHAR2(20);
BEGIN

  SELECT COUNT(*) INTO CNT  FROM USERSLIST WHERE USERNAME=UNAME AND PASSWORD=PWD;
  IF(CNT<>0) THEN
   RESULT:='VALID CREDENTIALS';
  ELSE
   RESULT:='INVALID CREDENTIALS';
  END IF;
 RETURN RESULT;
END FX_AUTHENTICATION;
```

Code in SErvice class
====================

```
/*CREATE OR REPLACE FUNCTION FX_AUTHENTICATION( UNAME IN VARCHAR2 , PWD IN VARCHAR2) RETURN VARCHAR2 AS
      CNT NUMBER(5);
      RESULT VARCHAR2(20);
    BEGIN

      SELECT COUNT(*) INTO CNT  FROM USERSLIST WHERE USERNAME=UNAME AND PASSWORD=PWD;
      IF(CNT<>0) THEN
       RESULT:='VALID CREDENTIALS';
      ELSE
       RESULT:='INVALID CREDENTIALS';
      END IF;
     RETURN RESULT;
    END FX_AUTHENTICATION;
    */

    @Override
    public String authenticate(String username, String password) {
        Session ses=em.unwrap(Session.class);
        String result=ses.doReturningWork(con->{
            try {
            //create CallableStatemene object
                CallableStatement cs=con.prepareCall("{?=call FX_AUTHENTICATION(?,?)}");
                //register  return parameter with JDBC type
                cs.registerOutParameter(1,Types.VARCHAR); //return param
                //set values to IN params
                cs.setString(2,username);
                cs.setString(3,password);
                //call PL/SQL function
                cs.execute();
                //get result from Return params
                String  output=cs.getString(1);
                return output;
            }//try
            catch(SQLException se) {
                se.printStackTrace();
                return null;
            }
        });
        return result;
    }//method
```

LAMDA Expression based anonymous inner class for ReturningWork(I)

Signature ses.doReturningWork(-) method

```
default <T> T doReturningWork(ReturningWork<T> work) throws HibernateException
```

Functional Interface, so we can pass LAMDA Expression based anonymous inner class.

Code in client app

```
System.out.println("-----------------------------------");
        System.out.println(service.authenticate("raja","rani"));
```

Calling PL/SQL procedure of mysql from Spring Data JPA Application
==

=>There are no cursors in PL/SQL programming of mysql , basically they are not required in mysql to hold bunch records given by the "SELECT" SQL query of PL/SQL of procedure/function

step1) makes sure that PL/SQL procedure is ready in mysql Db s/w

```
USE `ntsp713db`;
DROP procedure IF EXISTS `P_GET_EMPS_BY_CITIES`;

DELIMITER $$
USE `ntsp713db`$$
CREATE PROCEDURE P_GET_EMPS_BY_CITIES (IN city1 varchar(10), IN city2 varchar(10))
BEGIN
  SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE EADD IN(city1,city2);
END$$

DELIMITER ;
```

Launch mysql workbench--> expand ntsp713db --> go to Stored Procedure --> right click --> create new StoredPRocedure ---> write the code..

step2) change application.properties file to mysql Db s/w..

```
# for DataSoruce cfg
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntsp713db
spring.datasource.username=root
spring.datasource.password=root
# JPA -Hibernate Properties
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

step3) add mysql-connector-java-<ver>.jar dependency to file to pom.xml

```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
   <groupId>mysql</groupId>
   <artifactId>mysql-connector-java</artifactId>
   <version>8.0.22</version>
</dependency>
```

step4) Write code in service class

```
package com.nt.service;
import java.util.ArrayList;
import java.util.List;
import javax.persistence.EntityManager;
import javax.persistence.ParameterMode;
import javax.persistence.StoredProcedureQuery;
import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.nt.dto.EmployeeDTO;
import com.nt.entity.Employee;

@Service("empService")
public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService {
    @Autowired
    private  EntityManager em;

    /*CREATE DEFINER=`root`@`localhost` PROCEDURE `P_GET_EMPS_BY_CITIES`(IN city1 varchar(10), IN city2 varchar(10))
            BEGIN
              SELECT ENO,ENAME,EADD,ESALARY FROM  EMPLOYEE WHERE EADD IN(city1,city2);
            END
    */
    @Override
    public List<EmployeeDTO> fetchEmpsByCities(String city1, String city2) {
        //create StoredProcedureQuery object
        StoredProcedureQuery procedure=em.createStoredProcedureQuery("P_GET_EMPS_BY_CITIES", Employee.class);
        //register Prcodure  Parameters
        procedure.registerStoredProcedureParameter(1, String.class,ParameterMode.IN);
        procedure.registerStoredProcedureParameter(2, String.class,ParameterMode.IN);
        //set values to  IN params
        procedure.setParameter(1, city1);
        procedure.setParameter(2,city2);
        //execute PL/SQL procedure
        List<Employee> listEntities=procedure.getResultList();
        //convert  listEntities  to listDTO
        List<EmployeeDTO>  listDTO=new ArrayList();
        listEntities.forEach(entity->{
            EmployeeDTO dto=new EmployeeDTO();
            BeanUtils.copyProperties(entity, dto);
            listDTO.add(dto);
        });

        return  listDTO;
    }//method
}//class
```

step5) Write code in Client app

```
        //invoke the method
            service.fetchEmpsByCities("hyd", "delhi").forEach(System.out::println);
```

Working with Date Vlaues in Spring data JPA (By using java8 Date and time API (JODAtime API)

In Java 8 data and time api
LocalDate :: To construct date values
LocalTime :: To construct time values
LocalDateTime :: To construct date and time values

usecase:: Storing/Retrieving DOB, TOB , Bill Date ,Bill Time , DOJ , DOM ande etc..

oracle Date,time data types
----
date :: To store only date values
timestamp :: To store both date and time values

note: Here no time data type in oracle

mysql date ,time data types
----
date :: To store only date values
time :: To store only time values
datetime :: to store both date and time values

Code
=======

```
//CustomerInfo.java
package com.nt.entity;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

import lombok.Data;

@Entity
@Data
public class CustomerInfo {
    @Id
    @GeneratedValue
    private Integer custId;
    @Column(length = 20)
    private  String  custName;
    private  Float  billAmt;
    private  LocalDate  dob;
    private  LocalTime  tob;
    private LocalDateTime billDate;
}
```

```
//ICustomerInfoRepo.java
package com.nt.repository;

import org.springframework.data.jpa.repository.JpaRepository;

import com.nt.entity.CustomerInfo;

public interface ICustomerInfoRepo extends JpaRepository<CustomerInfo,Integer> {

}
```

```
//ICustomerInfoMgmtService.java
package com.nt.service;

import java.util.List;

import com.nt.dto.CustomerInfoDTO;

public interface ICustomerInfoMgmtService {
   public   String  registerCustomer(CustomerInfoDTO dto);
   public   List<CustomerInfoDTO>  fetchAllCustomerDetails();
}
```

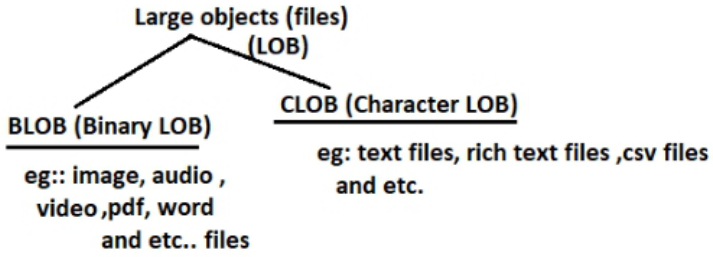
```
//CustomerInfoMgmtServiceImpl.java
package com.nt.service;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.nt.dto.CustomerInfoDTO;
import com.nt.entity.CustomerInfo;
import com.nt.repository.ICustomerInfoRepo;

@Service("customerService")
public class CustomerInfoMgmtServiceImpl implements ICustomerInfoMgmtService {
    @Autowired
    private   ICustomerInfoRepo  custRepo;

    @Override
    public String registerCustomer(CustomerInfoDTO dto) {
        //convert DTO  to Entity
        CustomerInfo entity=new CustomerInfo();
        BeanUtils.copyProperties(dto, entity);
        //use  repo
        int id=custRepo.save(entity).getCustId();
        return "Customer is registerd having Id ::"+id;
    }

    @Override
    public List<CustomerInfoDTO> fetchAllCustomerDetails() {
    //use repo
        List<CustomerInfo> listEntities=custRepo.findAll();
        //convert   listEntities to  listDTOs
        List<CustomerInfoDTO> listDTO=new ArrayList();
        listEntities.forEach(entity->{
            CustomerInfoDTO dto=new CustomerInfoDTO();
            BeanUtils.copyProperties(entity,dto);
            listDTO.add(dto);
        });

        return listDTO;
    }

}
```

```
//CustomerDTO.java
package com.nt.dto;

import java.io.Serializable;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.LocalTime;

import lombok.Data;

@Data
public class CustomerInfoDTO implements Serializable {
    private Integer custId;
    private  String  custName;
    private  Float  billAmt;
    private  LocalDate  dob;
    private  LocalTime  tob;
    private LocalDateTime billDate;


}
```

Code in client app
================

```
//prepare DTO
        CustomerInfoDTO dto=new CustomerInfoDTO();
        dto.setCustName("rajesh"); dto.setBillAmt(9000.0f);
        dto.setDob(LocalDate.of(1991, 10, 23));
        //dto.setTob(LocalDateTime.of(1990,10,23,17,20, 40));
        dto.setTob(LocalTime.of(18,20, 42));
        dto.setBillDate(LocalDateTime.now());
        try {
        System.out.println(service.registerCustomer(dto));
        }
        catch(DataAccessException dae) {
            dae.printStackTrace();
        }

    try {
        service.fetchAllCustomerDetails().forEach(System.out::println);
    }
    catch(DataAccessException dae) {
        dae.printStackTrace();
    }
```

Working with Large Objects (files)



=>if the App is standalone App then we store LOB files directly in DB table cols by taking the cols as the BLOB/CLOB colums
eg:: Standalone matrimony app, standalone Job Consultancy App and etc..
note:: Most of the popular DB s/ws give support for BLOB,CLOB data types like (oracle,mysql and etc..)

=> if the App is web application , then we store LOB files in the server machine file System and we write their locations to DB table cols as String values.
eg:: Shaddi.com, youtube.com , facebook.com and etc..

In Hibernate or Spring data JPA we need take byte[] ,char[] type proeprties in the entity class to inorder to map the properties of Entity class with Db table columns.

Entity class
==========

```
@Table
@Entity  @Data
public class JobSeeker{
  @Id
  private  Integer jsId;
  private  String jsName;
  private  String  qlfy;

  @Lob
  private  byte[]  photo;
  @Lob
  private   char[]  resume;
}
```

DB table in oracle
=======

```
JobSeeker
   |---> jsId  (pk) (n)
   |---> jsName  (vc2)
   |---> qlfy   (vc2)
   |---> photo   BLOB
   |---> resume   CLOB
```

Db table in mysql

```
JobSeeker
   |--->jsId (pk) (int)
   |--->jsName (vc)
   |--->qlfy  (vc)
   |---> photo  (BLOB)
   |---> resume   (LONGTEXT)
```

refer SpringDataJPAProj11-WorkingWithLOBs Project of GIT for complete code.

Association mapping in Spring Data JPA

=>Storing multiple the information about multiple entities /items in single dB table gives the following limitations

a) It creates data redundency(dupliication) Problem

b) Data maintainence becomes very complex

Problem::

StudentInfo (db table)

stid	stname	staddrs	mobileNo	type	provider
101	raja	hyd	9999999	office	airtel
101	raja	hyd	8888888	personal	vi
102	suresh	hyd	9888888	office	airtel
102	suresh	hyd	7888888	personal	jio
102	suresh	hyd	6788888	home	vi

Here Student Info is duplicated , more over storing the info of multiple entities in single table becomes very complex.

Solution1:: Take information in two different db tables (not recomanded)

Studnet_info (db table1)

stid(pk)	stname	staddrs
101	raja	hyd
102	suresh	hyd

phoneNumber_details (table2)

mobileNo(pk)	type	provider
9999999	office	airtel
8888888	personal	vi
9888888	office	airtel
7888888	personal	jio
6788888	home	vi

note:: Data duplication(Redundency Problem)is gone .. Data navigation problem is created.. i.e we can not access phone number details from student details and vice-versa.

Solution2: take two db tables and keep them relationship having FK (foreign key)
(Not Recomanded solution)

student_info (parent db table)

stId	stname	staddrs	FK mobileNo
101	raja	hyd	9999999
101	raja	hyd	8888888
102	suresh	hyd	9888888
102	suresh	hyd	7888888
102	suresh	hyd	6788888

Data Redundency

child
phoneNumber_details (table)

mobileNo(pk)	type	provider
9999999	office	airtel
8888888	personal	vi
9888888	office	airtel
7888888	personal	jio
6788888	home	vi

=>This solution solves the data navigation problem by using FK col but still continuing data redundency problem.

Solution3:: take two db tables and keep them relationship having FK (foreign key) in child db table
(Best solution)

parent
Studnet_info (db table:)

stid(pk)	stname	staddrs
101	raja	hyd
102	suresh	hyd

phoneNumber_details (child db table)

mobileNo(pk)	type	provider	stud_Id(FK)
9999999	office	airtel	101
8888888	personal	vi	101
9888888	office	airtel	102
7888888	personal	jio	102
6788888	home	vi	102

=>It is best solution becoz it solving data redundency problem and also solving data navigation problem

=>If we keep db tables in Association then we need to keep the relavent Entity class of ORM also in the Relationship/Association.

=> We keep db tables in relationship using FK colum support .. where as we keep Entity classes in relationship using collection / reference(or) Object type propeties i.e The way Db tables in relationship is totally different from that way java classes in relationshop But still we need to map Db tables of relationship with Entity classes of relationship by by using association mapping.

Association mapping or multiplicity is two types based on navigation

a) Uni-Directional (Parent to child or child to parent navigation is possible)

b) Bi-Directional (Parent to child and child to parent navigation is possible)

note:: One to Many , Many to One , One to One can be implemented either as uni-directional or as bi-directional association

note:: Many to Many Association is always bi-directional Association

=>Person and Passport (1 to 1 / 1 --1)
=> Person and DrivingLicense (1 to 1 / 1--1)
=> Department and Employees (1 to M / 1 --*)
=> User and PhoneNumbers (1 to M / 1 --*)
=> Employee and Department (M to 1 / * -- 1)
=>Vechile and Person (M to 1 / * --1)
=> Student and course (M to M / * -- *)
=> Student and Facuty (M to M / * -- *)
=> Company and Project (1 to M / 1 -- *)
=>Programmer and Project (M to 1/ * --1 in Most of the companies)
(M to M / * --* in some companies)
=>Movie and Actor (M to M / * --*)

There are two types Association mapping based on kind of properties we take in the Entity classes the

a) Collections bassed Association mapping
eg:: 1 to M or M to M

b) Non-collection based Association Mapping
eg: M to 1 and 1 to 1

OneToMany Bi-Directional Association

=>Here parent class should have collection property to hold one or more associated child class objs and child class should have parent class reference variable to hold the associated parent objs
=> It is one to many assocation from parent prospective and many to one association from child prospective

=> Example ::
Department- Employee (One to many)
|--->One Department can have multiple Employees (one to many) and
Multiple Employees can belong to one Department (many to one)

//Parent class

```
@Entity @Data @Table(name="DEPARTMENT_OTM")
public class Department implements Serializable{
 @Id
 @GeneratedValue
 private int deptNo;
 @Column(length= 20 )
 private String deptName;
 @Column(length= 20 )
 private String deptLocation;

 @OneToMany(targetEntity=Employee.class, cascade=CascadeType.ALL,fetch=FetchType.Lazy)
 @JoinColum(name="dept_no" , referenceColumn="deptNo")
 private Set<Employee> emps;

}
```

Child class

```
@Entity @Data
@Table(name="EMPLOYEE_OTM")
public class Employee implements Serializable{
 @Id
 @SquenceGenerator(name="gen1", ......)
 @GeneratedValue(generator="gen1",strategy=SEQEUNCE)
 private Integer eno;
 @Column(length=20)
 private String ename;
 @Column(length=20)
 private String eadd;
 private Float esalary;
 @ManyToOne(target Entity=Department.class, ....)
 @JoinColumn(name="dept_no", referencedColumn="detpno")
 private Department dept;
}
```

DEPARTMENT_OTM (parent db tabe)

deptno(pk)	deptName	deptLocation
1	Accounts	hyd

EMPLOYEE_OTM (child db table)

eno(pk)	ename	eadd	esalary	dept_no(fk)
1001	raja	hyd	80000	1
1002	ramesh	hyd	90000	1

join column

To unlock oracle Account

```
SQL> connect as Sysdba;
Enter user-name: system
Enter password:
Connected.
SQL>
SQL>
SQL>
SQL> alter user system account unlock;
User altered.
```

=>Using single FK column we can navigate from parent db table records to child db table records and vice-versa.. where as while desinging entity classes the parent class should have special property to navigate to child objs and similary child class should have special property to navigate to parent objs

cascade=CasdeType.ALL --> Indicates the non-select (insert,update,delete) persistence operations performed on the Entity object(parent/child) also takes palce on associated objects.
For example if we perform save object operation on parent obj then not only record will be inserted in parent db table(like Departmment) ..but the associated records will be inserted in child table (like Employee)

fetch=FetchType.LAZY --> Loads main objects (parent /child objs) normally but loads the associated child objects lazily on demand
=> For example if u r performing select operation using parent class.. then it loads parent objs normally but loads the associated child objs lazily on demand

mappedBy ---> In Bi-Directional Assoication , instead of placing FK column cfg using @JoinColumn in both parent and child classes, we can cfg only one side and we can sepcify the property name otherside by using "mappedBy" .. This can be done Many side (parent class) in one to many , any side in many to many and One to One.

```
@OneToMany(targetEntity = Employee.class,cascade = CascadeType.ALL, fetch =FetchType.LAZY ,mappedBy = "dept")
private Set<Employee> emps;
```

Child class special property name

One To many Assoication

=> fetch=FetchType.LAZY performs lazy loading i.e parent objs will be loaded normally but the associated (default) child objects will be loaded lazily on demand (i.e they will be loaded when we start accessing through parent objs)

=> fetch=FetchType.EAGER PErforms eager Loading i.e Along Parent objs loading the associated childs objs wil be loaded irrepspective of wheather u r using them or not.

=> In order to enable hibernate supplied lazy and eager loading options take the support @Lazyxxx annotations like @LazyToCollection in One to many Association.

```
@OneToMany(targetEntity = Employee.class,cascade = CascadeType.ALL, fetch =FetchType.LAZY ,mappedBy = "dept")
//@JoinColumn(name = "dept_no",referencedColumnName = "deptNo")
@LazyCollection(LazyCollectionOption.EXTRA)
private Set<Employee> emps;
```

TRUE
FALSE

FALSE :: performs eager loading
TRUE(default) :: Performs lazy loading but loads the child objects even for the aggragate operations performed collection like isEmpty() ,size() and etc.. (which is unneccessary)
EXTRA(best) :: Performs lazy loading but does not load the child objects for the aggragate operations performed collection like isEmpty() ,size() and etc.. (which is good)

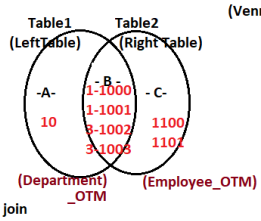
note:: if we specify JPA configuration and hibernate configurations with two different settings then the configurations done in Hibenrate settings will take place.

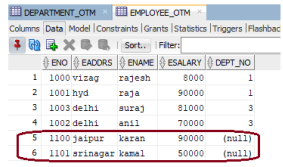
```
@ManyToOne(targetEntity = Department.class,cascade = CascadeType.ALL,fetch = FetchType.EAGER)
@JoinColumn(name="dept_no",referencedColumnName = "deptNo")
private Department dept;
```

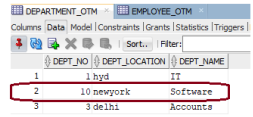
In Many to One Assoication the default fetch type is eager..

HQL /JPQL Joins
==========
=>These Queries are useful to get Data from multiple Db tables having implicit conditions..
=>For this , we need to keep DB tables and their related Entity classes in realationship
=> As of now HQL/JPQL supports 4 types of joins
a) inner join
b) Left join /Left outer join
c) Right join/ Right outer join
d) Full Join

(Venn diagarams)







inner join
=========
=>Gives only common data that is there in both left side db table and right side db table (-B-)

left join/left outer join
=========
=>Gives only common data that is there in both left side db table and right side db table (-B-) and also gives un common data that is there in left side db table (-A-)

right join/right outer join
=========
=>Gives only common data that is there in both left side db table and right side db table (-B-) and also gives un common data that is there in right side db table (-C-)

Full join
========
=>gives both common and uncommons data that is there in both left side and right side db tables (-A-, -B-,-C-)

Syntax of HQL join query (parent to child)
---------,child properties
select <parent prorperies> from <parent class> <join type> <HAS-A variable in parent class> <addional conditions>

//parent to child
---
select d.deptNo,d.deptName,d.deptLocation,e.eno,e.ename,e.eaddrs,e.esalary from Department d inner join d.emps e;
select d.deptNo,d.deptName,d.deptLocation,e.eno,e.ename,e.eaddrs,e.esalary from Department d left join d.emps e;
select d.deptNo,d.deptName,d.deptLocation,e.eno,e.ename,e.eaddrs,e.esalary from Department d right join d.emps e;
select d.deptNo,d.deptName,d.deptLocation,e.eno,e.ename,e.eaddrs,e.esalary from Department d full join d.emps e;

Syntax of HQL Join Query (child to parent)

select <child, parent classes properties> from <chid class> <join type> <HAS-A variable in child class> <addtional condition>
select e.eno,e.ename,e.eaddrs,e.esalary,d.deptNo,d.deptName,d.deptLocation from Employee e inner join e.dept d
select e.eno,e.ename,e.eaddrs,e.esalary,d.deptNo,d.deptName,d.deptLocation from Employee e left join e.dept d
select e.eno,e.ename,e.eaddrs,e.esalary,d.deptNo,d.deptName,d.deptLocation from Employee e right join e.dept d
select e.eno,e.ename,e.eaddrs,e.esalary,d.deptNo,d.deptName,d.deptLocation from Employee e full join e.dept d

Example code
==========
Repository interfaces
==========
```
public interface IDepartmentRepo extends JpaRepository<Department, Integer> {
	//@Query("select d.deptNo,d.deptName,d.deptLocation,e.eno,e.ename,e.eaddrs,e.esalary from Department d inner join d.emps e")
	//@Query("select d.deptNo,d.deptName,d.deptLocation,e.eno,e.ename,e.eaddrs,e.esalary from Department d left join d.emps e")
	//@Query("select d.deptNo,d.deptName,d.deptLocation,e.eno,e.ename,e.eaddrs,e.esalary from Department d right join d.emps e")
	@Query("select d.deptNo,d.deptName,d.deptLocation,e.eno,e.ename,e.eaddrs,e.esalary from Department d full join d.emps e")
	public List<Object[]> getJoinsDataParentToChild();
}

public interface IEmployeeRepo extends JpaRepository<Employee, Integer> {

	@Query("select e.eno,e.ename,e.eaddrs,e.esalary,d.deptNo,d.deptName,d.deptLocation from Employee e right join e.dept d")
  public List<Object[]> getJoinsDataChildToParent();

}
```
Service interface
```
public interface ICompanyMgmtService {
  public List<Object[]> fetchJoinsDataParentToChild();
  public List<Object[]> fetchJoinsDataChildToParent();

}
```
service impl class
==========
```
@Service("compService")
public class CompanyMgmtServiceImpl implements ICompanyMgmtService {
	@Autowired
	private IDepartmentRepo deptRepo;
	@Autowired
	private IEmployeeRepo empRepo;

	@Override
	public List<Object[]> fetchJoinsDataParentToChild() {
		List<Object[]> list=deptRepo.getJoinsDataParentToChild();
		return list;
	}

	@Override
	public List<Object[]> fetchJoinsDataChildToParent() {
		List<Object[]> list=empRepo.getJoinsDataChildToParent();
		return list;
	}

}//class
```
main class
==========
```
@SpringBootApplication
public class SpringDataJpa12AssoicationMappingOneToManyApplication {

	public static void main(String[] args) {
		//get IOC container
		ApplicationContext ctx=SpringApplication.run(SpringDataJpa12AssoicationMappingOneToManyApplication.class, args);
		//get Service class object
		ICompanyMgmtService service=ctx.getBean("compService",ICompanyMgmtService.class);
		//invoke methods
		/*service.fetchJoinsDataParentToChild().forEach(row->{
			for(Object value:row) {
				System.out.print(value+"   ");
			}
			System.out.println();
		});*/
		System.out.println("-----------------------------------------------");
		service.fetchJoinsDataChildToParent().forEach(row->{
			for(Object val:row) {
				System.out.print(val+"   ");
			}
			System.out.println();
		});

		//close comtainer
		((ConfigurableApplicationContext) ctx).close();
	}//main
}//class
```

Spring Data MangoDB
===================

=> Spring Data provides unified model to work with both SQL and NoSQL Db s/ws...

=> SQL Db s/ws are oracle, postgreSQL, mysql and etc.. (Db tables based)
=> No SQL Db s/ws are MangoDB , cassendra, couchbase, neo4j and etc.. (key-value based,
document based,
graph based and etc..)

Difference between SQL and NoSQL

Parameter	SQL	NOSQL
Definition	SQL databases are primarily called RDBMS or Relational Databases	NoSQL databases are primarily called as Non-relational or distributed database
Design for	Traditional RDBMS uses SQL syntax and queries to analyze and get the data for further insights. They are used for OLAP systems.	NoSQL database system consists of various kind of database technologies.These databases were developed in response to the demands presented for the development of the modern application.
Query Language	Structured query language (SQL)	No declarative query language
Type	SQL databases are table based databases	NoSQL databases can be document based, key-value pairs, graph databases
Schema	SQL databases have a predefined schema	NoSQL databases use dynamic schema for unstructured data.
Ability to scale	SQL databases are vertically scalable	NoSQL databases are horizontally scalable
Examples	Oracle, Postgres, and MS-SQL.	MongoDB, Redis, , Neo4j, Cassandra, Hbase.
Best suited for	An ideal choice for the complex query intensive environment.	It is not good fit complex queries.
Hierarchical data storage	SQL databases are not suitable for hierarchical data storage.	More suitable for the hierarchical data store as it supports key-value pair method.
Variations	One type with minor variations.	Many different types which include key-value stores, document databases, and graph databases.
Development Year	It was developed in the 1970s to deal with issues with flat file storage	Developed in the late 2000s to overcome issues and limitations of SQL databases.
Open-source	A mix of open-source like Postgres & MySQL, and commercial like Oracle Database.	Open-source
Consistency	It should be configured for strong consistency.	It depends on DBMS as some offers strong consistency like MongoDB, whereas others offer only offers eventual consistency, like Cassandra.
Best Used for	RDBMS database is the right option for solving ACID problems.	NoSQL is a best used for solving data availability problems
Importance	It should be used when data validity is super important	Use when it's more important to have fast data than correct data
Best option	When you need to support dynamic queries	Use when you need to scale based on changing requirements
Hardware	Specialized DB hardware (Oracle Exadata, etc.)	Commodity hardware
Network	Highly available network (Infiniband, Fabric Path, etc.)	Commodity network (Ethernet, etc.)
Storage Type	Highly Available Storage (SAN, RAID, etc.)	Commodity drives storage (standard HDDs, JBOD)
Best features	Cross-platform support, Secure and free	Easy to use, High performance, and Flexible tool.
Top Companies Using	Hootsuite, CircleCI, Gauges	Airbnb, Uber, Kickstarter
ACID vs. BASE Model	ACID(Atomicity, Consistency, Isolation, and Durability) is a standard for RDBMS	Base (Basically Available, Soft state, Eventually Consistent) is a model of many NoSQL systems

MongoDB
=========

=> It is Docucment based NO SQL Db s/w.
=> It internally maintains the data in the form of JSON documents
JSON :: Java Script Object Notation..

Examples of JSON Data...

```
{
    "Name": "Kim",
    "address": {
    "streetAddress": "1st Street",
    "city": "Seoul",
    "postalCode": 1112
    },
    "phoneNumbers": [
    "111-222-333",
    "555-666-777"
    ]
}
```

```
{
  "emp_details": [
    {
      "emp_name": "Shubham",
      "email": "ksingh.shubh@gmail.com",
      "job_profile": "intern"
    },
    {
      "emp_name": "Gaurav",
      "email": "gaurav.singh@gmail.com",
      "job_profile": "developer"
    },
    {
      "emp_name": "Nikhil",
      "email": "nikhil@geeksforgeeks.org"
      "job_profile": "Full Time"
    }
  ]
}
```

```
{
  "api": {
    "title": "Example API",
    "links": {
      "author": "mailto:api-admin@example.com",
      "describedBy": "https://example.com/api-docs/"
    }
  }
  "resources": {
    "tag:me@example.com,2016:widgets": {
      "href": "/widgets/"
    },
    "tag:me@example.com,2016:widget": {
      "hrefTemplate": "/widgets/{widget_id}",
      "hrefVars": {
        "widget_id": "https://example.org/param/widget"
      },
      "hints": {
        "allow": ["GET", "PUT", "DELETE", "PATCH"],
        "formats": {
          "application/json": {}
        },
        "acceptPatch": ["application/json-patch+json"],
        "acceptRanges": ["bytes"]
      }
    }
  }
}
```

MangoDB s/w installation
======================

MongoDB download::
https://www.mongodb.com/try/download/community --> select current version (4.4.3) --->
select widows --> select msi ---> download.
⇩
Install like any other
windows software..

note:: we have multiple MongoDB gui Tools or clients s/ws like compass, Robot3T and etc..

To start MongoDB software ::
Go to C:\Program Files\MongoDB\Server\4.4\bin a and use mango.exe file...

```
SQL terminlogies
----------------------
 Physical DB s/w (like oracle)
     |---->Logical DB1
     |          |----> db table1  (cols and rows)
                |----> db table2  (cols and rows)
                |----> db table3  (cols and rows)
     |---> Logical DB2
```

```
NoSQL MongoDB terminologies
-------------------------------------------
 Physical Db s/w (like MongoDB)
    |-----> Logical DB1
    |          |---->Collection1 (document1, document2,.. (json docs)
    |          |---->Collection2 (document1, document2,.. (json docs)
    |----->Logical DB2
```

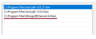
=>When we insert document to collection the mongoDb dynamically generateds unique ID as Hexadecimal number

Binary --> (Base2) :: 0 1
Decimal --> (Base10) :: 0-9
Octal --> (Base8) :: 0-7
Hexa ---> (Base16) --> 0-9 , a-f

Mongodb Shell commands
----

1) add <MongoDB_home>\bin folder to PATH env... variable

This PC --->properties ---> advanced system setttings --> env.. variables -->system variable --->
name :: PATH value::



--> ok --> ok -->ok

2) Open MongoDB Shell using mongo.exe file from any command prompt..

cmd> mongo (from any location)

3) To list all logical Dbs..

cmd> show dbs

4) To create new Logical DB
cmd> use ntsp713DB
switched to db ntsp713DB

5) To know current logical DB
cmd> db
ntsp713DB

6)To create collection with one document..
```
cmd> db.customer.insertOne({cno:1001,cname:"raja",cadd:"hyd",billAmt:89000.0})
{
      "acknowledged" : true,
      "insertedId" : ObjectId("60078de7de61029897aadf28")
}
```

```
cmd> db.customer.insertOne({cno:1002,cname:"rajesh"})
{
      "acknowledged" : true,
      "insertedId" : ObjectId("60078eb8de61029897aadf29")
}
cmd> db.customer.insertOne({cno:1003,cname:"rajesh",mobileNo:987677445})
{
      "acknowledged" : true,
      "insertedId" : ObjectId("60078edfde61029897aadf2a")
}
```

7) To list out docummnents of the collection
```
cmd> db.customer.find()
{ "_id" : ObjectId("60078de7de61029897aadf28"), "cno" : 1001, "cname" : "raja", "cadd" : "hyd", "billAmt" : 89

cmd> db.customer.find().pretty()
{
      "_id" : ObjectId("60078de7de61029897aadf28"),
      "cno" : 1001,
      "cname" : "raja",
      "cadd" : "hyd",
      "billAmt" : 89000
}
```

8) To insert many documents at once to a Collection
```
> db.customer.insertMany([{cno:567,cname:"ramesh",cadd:"hyd"},{cno:789,cname:"rakesh"}])
                               doc1                             doc2
{
      "acknowledged" : true,
      "insertedIds" : [
            ObjectId("60078fb1de61029897aadf2b"),
            ObjectId("60078fb1de61029897aadf2c")
      ]
}
```
for reference ::
https://docs.mongodb.com/manual/tutorial/insert-documents/

9) To insert document to a collection with array values..
```
> db.customer.insertOne({cno:1003,cname:"rajesh",mobileNo:[987671245,6788886777]})
{
      "acknowledged" : true,
      "insertedId" : ObjectId("6007911ade61029897aadf2d")
}
```

10) To find docs of a collection with condition
```
> db.customer.find({cadd:"hyd"}).pretty()
{
      "_id" : ObjectId("60078de7de61029897aadf28"),
      "cno" : 1001,
      "cname" : "raja",
      "cadd" : "hyd",
      "billAmt" : 89000
}
{
      "_id" : ObjectId("60078fb1de61029897aadf2b"),
      "cno" : 567,
      "cname" : "ramesh",
      "cadd" : "hyd"
}
```

```
> db.customer.find({cadd:"hyd",cname:"raja"}).pretty()
{
      "_id" : ObjectId("60078de7de61029897aadf28"),
      "cno" : 1001,
      "cname" : "raja",
      "cadd" : "hyd",
      "billAmt" : 89000
}
```

11) To delete mongodb document
```
> db.customer.remove({cno:1001 })
WriteResult({ "nRemoved" : 1 })

> db.customer.deleteOne({cno:1002})
{ "acknowledged" : true, "deletedCount" : 1 }

> db.customer.deleteMany({cno:1003})
{ "acknowledged" : true, "deletedCount" : 2 }
> db.customer.find().pretty()
{
      "_id" : ObjectId("60078fb1de61029897aadf2b"),
      "cno" : 567,
      "cname" : "ramesh",
      "cadd" : "hyd"
}
{
      "_id" : ObjectId("60078fb1de61029897aadf2c"),
      "cno" : 789,
      "cname" : "rakesh"
}
```
_id key holds dynamically generated unique value as pk VALUE.

12) To remove collection
```
> db.customer.drop()
true
```

13) To list all collections
>> show collections

14) To switch to other logical DB
>use CollegeDB

=>Spring Data provides two approaches to perfrom CURD Opertions and other operations on MongoDB
a) Using MongoRepository (more recomanded)
b) Using MongoTemplate

Working MongoRepository
======================

Repository(I)
↑
CrudRepository(I)
↑
PagingAndSortingRepository(I)
↑
MongoRepository(I)

Spring Boot allows to work with
a) Embedded InMemory MongoDB s/w (only testing)
b) External MongoDB s/w

Example App using MangoRepository
==============================

step1) create Spring Boot starter Project having mongodb , lomobok api dependencies..

step2) develop application.properties
```
spring.data.mongodb.host=localhost
spring.data.mongodb.database=ntsp713DB
spring.data.mongodb.port=27017
```

step3) Develope Document and DTO classes
```java
//Customer.java
package com.nt.document;
import org.springframework.data.annotation.Id;
import org.springframework.data.mongodb.core.mapping.Document;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.NonNull;
import lombok.RequiredArgsConstructor;
@Document
@Data
@RequiredArgsConstructor
@NoArgsConstructor
public class Customer {
    @Id
    private String id;
    @NonNull
    private Integer cno;
    @NonNull
    private String cname;
    @NonNull
    private  Float billAmt;

}
```

```java
//CustomerDTO.java
package com.nt.dto;

import java.io.Serializable;


import lombok.AllArgsConstructor;
import lombok.Data;

@Data
@AllArgsConstructor
public class CustomerDTO implements Serializable {
        private Integer cno;
        private String cname;
        private  Float billAmt;

}
```

step3) Develope Repository Interface..
```java
CustomerRepo.java
package com.nt.repository;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.nt.document.Customer;
public interface CustomerRepo extends MongoRepository<Customer, String> {

}
```

step4) Develop service Interface and Service Impl classes..
```java
//ICustomerMgmtService.java
package com.nt.service;

import com.nt.dto.CustomerDTO;
public interface ICustomerMgmtService {
   public  String  registerCustomer(CustomerDTO dto);
}
```

```java
// CustomerMgmtServiceImpl.java
package com.nt.service;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import com.nt.document.Customer;
import com.nt.dto.CustomerDTO;
import com.nt.repository.CustomerRepo;

@Service("custService")
public class CustomerMgmtServiceImpl implements ICustomerMgmtService {
    @Autowired
    private CustomerRepo  custRepo;

    @Override
    public String registerCustomer(CustomerDTO dto) {
        //convert dto to document obj
        Customer doc=new Customer();
        BeanUtils.copyProperties(dto, doc);
        //save document
        doc=custRepo.insert(doc);
        return "Document is saved having cno:"+doc.getCno();
    }

}
```

step5) Develop the Test class...
```java
//SpringDataMongoDb14CurdOperationsApplication.java
package com.nt;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.ConfigurableApplicationContext;
import com.nt.dto.CustomerDTO;
import com.nt.service.ICustomerMgmtService;
@SpringBootApplication
public class SpringDataMongoDb14CurdOperationsApplication {
    public static void main(String[] args) {
        //get IOC container
        ApplicationContext ctx=SpringApplication.run(SpringDataMongoDb14CurdOperationsApplication.class, args);
        //get Service obj
        ICustomerMgmtService service=ctx.getBean("custService",ICustomerMgmtService.class);
        System.out.println(service.registerCustomer(new CustomerDTO(901,"Rajesh",6789.77f)));

        //close container
        ((ConfigurableApplicationContext) ctx).close();
    }

}
```

step6) run the eapp as Spring boot app

step7) check in the MongoDB
```
> use ntsp713DB
switched to db ntsp713DB
> show collections
customer
> db.customer.find().pretty()
{
      "_id" : ObjectId("60079c2341e4554786795a8e"),
      "cno" : 901,
      "cname" : "Rajesh",
      "billAmt" : 6789.77001953125,
      "_class" : "com.nt.document.Customer"
}
>
```

What is difference b/w save(-) and insert(-) methods of MonogRepository?

=>save(-) is inherited method of CrudRepository(I) and it can perform save or update document operation (if id is not available then it performs save operation otherwise it performs update operation)

=>insert(-) is direct method of MongoRepository and it can perform only save document operation..

=>while working spring data mongoDB for save document operation , we can let mongoDb generating id value or we can set manually or we can use custom Generator to genere the id value (ObjectId)

=>if the Document object properties can not maintain unique values then take @Id property as String the prooperty and let MongoDB generating unique String id value..with setting value to that property or u can use custom ID generator to generate the unique id value..

```
@Document
@Data
@RequiredArgsConstructor
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
	@Id
	private String id;
	private Integer cno;
	private String cname;
	private Float billAmt;
	private String cadd;

}
```

if the Document object propety can maintain unique values .. then take the property itself as the @Id property..

```
@Document
@Data
@RequiredArgsConstructor
@NoArgsConstructor
@AllArgsConstructor
public class Customer {
	@NonNull
	@Id
	private Integer cno;
	private String cname;
	private Float billAmt;
	private String cadd;

}
```

note:: No built-in ID generators are available in in MongoDB..

MongoDB GUI Clients
a) Robot3T (for beginers)
b) Studio 3T (for proffessionals)
c) Compass (heavy software)

Procedure to work with Robot3T
==========

step1) Dowload Robot3T software
https://robomongo.org/download ---> Download Robot3T only ---> fillup the details ---->
select exe file to download (robo3t-1.4.2-windows-x86_64-8650949.exe) .

step2) Install Robot3T software .. (like any other windows software installation)

step3) Launch robot3T and establish the connection
File -> connect -->create ---> submit the details...
connection name :: con1
host name :: localhost
port number :: 27017
-->test -->connect ---> save.

step4) perform all the operations like creating logical DB , creating collection and adding/modifying/deleting docs and etc..

Procedure to add user MongoDB logical DB
==========
Expand NTSP713DB in Robot3T --->users (right click) ---> add user --->
username:: testuser
password :: testuser
select read , readwrite roles ---> save.

Connecting to LogicalDB (NTSP713DB) having username,password from Robot3T

=>Delete previous connection "con1" by using right click on con1 --> remove.
=> Create new connection with authentication..
File --> connection --> create -->

connection name :: con1
host name :: localhost
port number :: 27017 ---> Authentication tab --->
Database :: ntsp713DB (logical DB name)
username :: testuser
password :: testuser
--> test -->connect ---> save.

=Once we got username,password authentication.. we need to add
the following addtional entries in application.properties

```
spring.data.mongodb.host=localhost
spring.data.mongodb.database=ntsp713DB
spring.data.mongodb.port=27017
spring.data.mongodb.username=testuser
spring.data.mongodb.password=testuser
```

While wokring MongoRepository we can also have custom finder methods like spring data jpa repositories like findByXxxx(...) .

In Repository

```
package com.nt.repository;

import java.util.List;
import org.springframework.data.mongodb.repository.MongoRepository;
import com.nt.document.Customer;

public interface CustomerRepo extends MongoRepository<Customer, Integer> {
    public List<Customer> findByCadd(String cadd);
    public List<Customer> findByCnoBetween(int start,int end);
}
```

In service interface

```
public interface ICustomerMgmtService {
    public List<CustomerDTO> fetchCustomersByCadd(String cadd);
    public List<CustomerDTO> fetchCustomersByCnoRange(int start,int end);

}
```

In service impl class

```
@Service("custService")
public class CustomerMgmtServiceImpl implements ICustomerMgmtService {
    @Autowired
    private CustomerRepo custRepo;


    @Override
    public List<CustomerDTO> fetchCustomersByCadd(String cadd) {
        //use repo
        List<Customer> listDocs=custRepo.findByCadd(cadd);
        //convert listDocs to listDTO
        List<CustomerDTO> listDTO=new ArrayList();
        listDocs.forEach(doc->{
          CustomerDTO dto=new CustomerDTO();
          BeanUtils.copyProperties(doc, dto);
          listDTO.add(dto);
        });

        return listDTO;
    }

    @Override
    public List<CustomerDTO> fetchCustomersByCnoRange(int start, int end) {
        //use repo
            List<Customer> listDocs=custRepo.findByCnoBetween(start, end);
            //convert listDocs to listDTO
            List<CustomerDTO> listDTO=new ArrayList();
            listDocs.forEach(doc->{
              CustomerDTO dto=new CustomerDTO();
              BeanUtils.copyProperties(doc, dto);
              listDTO.add(dto);
            });

            return listDTO;
    }


}//class
```

In client App
```
service.fetchCustomersByCadd("hyd").forEach(System.out::println);
service.fetchCustomersByCnoRange(5000, 100000).forEach(System.out::println);
```

Special in types in MongoDB
==========
a) array/list/set
b) map/properties
c) HAS -A property
d) Relationships (Non-Collection) (one to one and Many to one)
e) Realtionships (Collection) (One to many and many to many)

=>In MongoDB array/list/set will be treated same :: [<val1>,<val2>,<val3>,...]
=>In MongoDB map/propeties will be treated same :: {<key1>:<val1>,<key2>:<val2>,<key3>:<val3>,...}

example code
==========
```
//Document class
@Document
@Data
@AllArgsConstructor
@NoArgsConstructor
public class Student {
    @Id
    private String id;
    private int sno;
    private String sname;
    private Integer marks[];
    private List<String> subjects;
    private Set<Long> phoneNumbers;
    private Map<String,Integer> ageDetails;
    private Properties batchesInfo;
}
```

```
Repository interface
---
public interface StudentRepo extends MongoRepository<Student, String> {

}
```

```
Client App
---
@SpringBootApplication
public class SpringDataMongoDb16SpecialProperitesApplication {

    public static void main(String[] args) {
        //get IOC container
  ApplicationContext ctx=  SpringApplication.run(SpringDataMongoDb16SpecialProperitesApplication.class, args);
  //get Repo object
  StudentRepo repo=ctx.getBean(StudentRepo.class);
  //save document
  Properties batchesInfo=new Properties();
  batchesInfo.put("Cjava","11am"); batchesInfo.put("advjava","4pm");
  repo.save(new Student(IDGenerator.generate(),
                        new Random().nextInt(1000),
                        "rajesh",
                        new Integer[] {56,88,99},
                        List.of("C","JAVA","spring"),
                        Set.of(9999991L,888888881L,77777771L),
                        Map.of("rajesh",30,"suresh",25,"ramesh",45),
                        batchesInfo
                        ));


  //close container
  ((ConfigurableApplicationContext) ctx).close();

    }

}
```

Document in MongoDB
==========
```
{
    "_id" : "52e5969bdf",
    "sno" : 73,
    "sname" : "rajesh",
    "marks" : [
        56,
        88,
        99
    ],
    "subjects" : [
        "C",
        "JAVA",
        "spring"
    ],
    "phoneNumbers" : [
        NumberLong(7777777),
        NumberLong(88888888),
        NumberLong(999999)
    ],
    "ageDetails" : {
        "suresh" : 25,
        "ramesh" : 45,
        "rajesh" : 30
    },
    "batchesInfo" : {
        "Cjava" : "11am",
        "advjava" : "4pm"
    },
    "_class" : "com.nt.document.Student"
}
```

MongoDB special properties for Assoications

a) HAS-A property (Non-Collection type)
   To build one to one and many to one association
b) HAS-A property (Collection type) (List/Map)
   To build one to many and many to many association

One to One Assoication

```
//parent class
public class Person{
  private int pid;
  private String pname;
  private String padd;
  private DrivingLicense license;
  //setters && getters
   ....
   ...
}
```

```
//DrivingLicense (chid class)
public class DrivingLicense{
   private int lid;
   private String type;
   private LocalDate expiryDate;
   //setters && getters
    ....
    ...
}
```

JSON doc of MongDOB

```
{
 ... //person obj properties (parent properties)
 ...
 license:{
        ...
        ... DrivingLicense object properties (child properies)
        ..
       }
}
```

Example code
==========

```
//parent class
@Document
@Data
@NoArgsConstructor
@AllArgsConstructor
public class Person {
    @Id
    private Integer pid;
    private String pname;
    private String paddrs;
    private DrivingLicense license;
}
```

```
//child class
@Document
@Data
@AllArgsConstructor
@NoArgsConstructor
public class DrivingLicense {
    @Id
    private Integer lid;
    private String type;
    private LocalDate expiryDate;

}
```

Repository Interface

```
public interface IPersonRepo extends MongoRepository<Person, Integer> {

}
```

main class

```
@SpringBootApplication
public class SpringDataMongoDb17SpecialPropertiesHasAOneToOneApplication {

    public static void main(String[] args) {
        //get IOC container
        ApplicationContext ctx=SpringApplication.run(SpringDataMongoDb17SpecialPropertiesHasAOneToOneApplication.class, args);
        //get Repository object
        IPersonRepo repo=ctx.getBean(IPersonRepo.class);
        //inovke the method
        DrivingLicense license=new DrivingLicense(90001, "4-wheer",LocalDate.of(2040, 10, 12));
        Person per=new Person(101,"rakesh","hyd",license);
          repo.save(per);
          System.out.println("object is saved...");
    }
}
```

Resultant JSON document in MongoDB

```
{
   "_id" : 101,
   "pname" : "rakesh",
   "paddrs" : "hyd",
   "license" : {
      "_id" : 90001,
      "type" : "4-wheer",
      "expiryDate" : ISODate("2040-10-11T18:30:00.000Z")
   },
   "_class" : "com.nt.doc.Person"
}
```

---

b) HAS-A property (Collection type) (List/Map)

To build one to many and many to many association

if the HAS-A property type array/list/set collection then document look like this

```
{
  ... //parent obj
  ...    properites
 HasAobj :[{.....},{....},{....},....]
 }
          represnets array/List/Set of child objs
```

if the HAS-A property type Map collection then document look like this

```
{
 ... //parent obj
 ...       properties
 ...
 hasAobj: { key1:{.....} ,key2:{....} ,key3: {...},....}
}
          represents map of child objs
```

Example App on One to many Assoication
==========

refer SpringDataMongoDB18-SpecialProperties-HAS-AOneToMany

```
{
   "_id" : 4561,
   "pname" : "rakesh",
   "paddrs" : "hyd",
   "visas" : [
      {
         "visaNo" : NumberLong(567718),
         "country" : "CANDADA",
         "expiryDate" : ISODate("2031-10-09T18:30:00.000Z")
      },
      {
         "visaNo" : NumberLong(567778),
         "country" : "USA",
         "expiryDate" : ISODate("2030-10-09T18:30:00.000Z")
      }
   ],
   "accounts" : {
      "account2" : {
         "_id" : NumberLong(2465654),
         "type" : "current",
         "balance" : 170000.0
      },
      "account1" : {
         "_id" : NumberLong(1465654),
         "type" : "savings",
         "balance" : 70000.0
      },
      "account3" : {
         "_id" : NumberLong(3465654),
         "type" : "savings",
         "balance" : 270000.0
      }
   },
   "_class" : "com.nt.doc.PersonInfo"
}
```

MongoDB @Query method with Projections
==========

=>By default @Query placed MongoDB app selects all the fields of the Document class
=> Use "fields" attribute of @Query annotation having field name with 0 or 1
syntax:: @Query(fields="{property:0/1, property:0/1 }")
=> 1 indicates involve the field/variable/property in the select query.
=> 0 indicates do not involve the field/variable/property in the select query.
For all properties default value is 0 , but for @Id property the default value is 1.

=>"value" attribute of @Query is useful to specify the where condition clauses..

eg1: @Query(value="{cadd:?0}",
fields="{cname:1,billAmt:1}")

is equal to "SELECT CNO,CNAME,BILLAMT FROM CUSTOMER WHERE CADD=?" (SQL)
(By Default @Id property will be seleted becoz its default value is 1)

eg2: @Query(value="{cadd:?0}",
fields="{cno:0,cname:1,billAmt:1}")

is equal to "SELECT CNAME,BILLAMT FROM CUSTOMER WHERE CADD=?" (SQL)

eg3:: To get all fileds/property values

@Query(value="{cadd:?0}",
fields="{ }")
or
@Query(value="{cadd:?0}")

(special case)

is equal to "SELECT * FROM CUSTOMER WHERE CADD=?" (SQL)

eg4:: To use multiple fields in where clause..
@Query(value="{cadd:?0,cname:?1}")

is equal to "SELECT * FROM CUSTOMER WHERE CADD=? AND CNAME=?" (SQL)

Requalar expression In MongoDB @Query methods

	In SQL	In MongoDB
Starting with given data	data%	^data
Ending with given data	%data	data$
Having given data	%data%	data or ^input$ (It is not working in few versions of MongoDB)

For Documentation..
https://docs.mongodb.com/manual/tutorial/query-documents/

eg5: @Query(value="{cadd:{$regex:?0}}")
public List<Customer> getRegData(String addrs);

while calling method from Client app
List<Customer> list=repo.getData("^h"); --> gives all docs whose cadd starts with "h"
List<Customer> list=repo.getData("h$"); --> gives all docs whose cadd ends with "h"

Example code

```
public interface CustomerRepo extends MongoRepository<Customer, Integer> {

    //@Query(fields="{cname:1,billAmt:1}",value="{cadd:?0}")
    @Query(fields="{cno:0,cname:1,billAmt:1}",value="{cadd:?0}")
    public List<Object[]> getData(String addrs);

    //@Query(fields="{}",value="{cadd:?0}")
     @Query(value="{cadd:?0}")
    public  List<Customer> getAllData(String addrs);

     @Query(value="{cadd:?0,cname:?1}")
    public  List<Customer> getAllData1(String addrs,String name);


     @Query(value="{cadd:{$regex:?0}}")
     public  List<Customer> getRegData(String addrsData);

   @Query(value="{cno:{$gt:?0},cno:{$lt:?1}}")
   public  List<Customer> getDataByCnoRange(int start, int end);

   @Query(value="{ $or: [ { cname:{$regex:?0 }, { cadd: ?1}  ] }")
   public  List<Customer> getDataByOrCond(String cnameInitChars,String cadd);
}
```

```
@SpringBootApplication
public class SpringDataMongoDb14CurdOperationsApplication {

    public static void main(String[] args) {
        //get IOC container
        ApplicationContext ctx=SpringApplication.run(SpringDataMongoDb14CurdOperationsApplication.class, args);
        //get Service obj
        CustomerRepo repo=ctx.getBean(CustomerRepo.class);
        //invoke methods
        /*List<Object[]> list=repo.getData("hyd");
        list.forEach(row->{
            for(Object val:row) {
                System.out.print(val+"  ");
            }
            System.out.println();
        });
        */
        System.out.println("..................................");
        /*List<Customer> list=repo.getAllData("hyd");
        list.forEach(cust->{
            System.out.println(cust);
        });*/

        /*System.out.println("..................................");
        List<Customer> list=repo.getAllData1("hyd","suresh");
        list.forEach(cust->{
            System.out.println(cust);
        });*/

        System.out.println("..................................");
        //List<Customer> list=repo.getRegData("^h");  // cadd starting with h
        //List<Customer> list=repo.getRegData("g$");  // cadd end with g
        /*      List<Customer> list=repo.getRegData("z");  // cadd containing  y
                list.forEach(cust->{
                    System.out.println(cust);
                });
        */

    /*  List<Customer> list=repo.getDataByCnoRange(3000,10000);
        list.forEach(System.out::println); */

        List<Customer> list=repo.getDataByOrCond("^s","hyd");
        list.forEach(System.out::println);


        //close container
        ((ConfigurableApplicationContext) ctx).close();
    }

}
```