Soft Deletion in spring MVC and spring data JPA

=> Deleting record perminently from db table is called hard deletion
eg: Ticket cancellation , Booking cancellation and etc..
=> marking the record as DELETED record by changing the status of the record and making that not participating in CURD oparations is called SoftDeletion
eg: BankAccount closing , Employee resignation and etc..

=> we have hibernate Annotations called @SQLXxx annotation which allows to configure custom SQL Queries for standard SQL opeartions like insert,update,delete . (@SQLInsert , @SQLDelete , @SQLUpdate)
@SQLDelete annotation is useful to cfg Custom Query for the standard delete operation . This is very to cfg UPDATE SQL query of soft deletion to mark the record as the DELETED record for standard delete operation.

note:Using @SQLXxx annotations we can change the Standard SQL queries of Repositiry methods to our choice custom SQL Queries

@Where annotation can be used to specify implicit condtion that should be applied on every query that executes..

@SQLXxx annotations are hibernate annotations

Steps for soft Deletion

a) create db table having status column with "active" value for all records..

Using SQL browser of SQL developer
=> create table BOOT_EMP as select empno,ename,job,sal,deptno from emp;

** ADD STATUS colum to db table BOOT_EMP

=> UPDATE BOOT_EMP SET STATUS='active' //Commit the records commit; (to commit the copied records)

b) Add @SQLDELETE Annotation specifying update query of softdeletion

@SQLDelete(sql="UPDATE BOOT_EMP SET STATUS='INACTIVE' WHERE EMPNO=?")
(write on the top of Entity class)

c) add @Where annotation specify condition to eleminate softely deleted records from regular CURD Operations. (avoid)

Entity class
=========

```
package com.nt.model;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;

import org.hibernate.annotations.SQLDelete;
import org.hibernate.annotations.Where;

import lombok.Data;

@Entity
@Table(name="boot_emp")
@Data
@SQLDelete(sql = "UPDATE BOOT_EMP SET STATUS='deleted' WHERE EMPNO=?")
@Where(clause = "STATUS <> 'deleted' ")
public class Employee {
    @Id
    @SequenceGenerator(name = "gen1",sequenceName = "emp_no_seq1",initialValue =3000,allocationSize = 1 )
    @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
    private Integer empno;
    private String ename;
    private String job;
    private Float sal;
    private Integer deptno=10;
    private String status="active";
}
```

hibernate annotations

note:: original repo.delete(-) ,deleteXxx() methods generates delete SQL Queries for deleting the records .. By using @SQLUpdate we can change those standard SQL Queries with our choice queries using which we can perform soft deletion activity

note:: @Where annotation is deprecated from hibernate 6.2 version (spring boot 3.2) ,So use @SqlRestriction as the alternate @SqlRestriction("STATUS <> 'deleted'")

Not equal to

=>Using Repository methods like delete(-) we can do only soft deletion on this db table using this Entity class..
=>To perform hard deletion on this db table we can use @Query methods of repository eithe with JPQL or Native SQL

softly deleted records

d) Run the Application..

=> JPA is the software specification giving rules and guidelines to create set of ORM frameworks like Hibernate , IBatis , Eclipse Link and etc..

if Entity class of spring data jpa is having @Version property then we need to provide that version related condition in the @SQLDelete annotation that is related to soft deletion activitity

```
@Entity
@Table(name="JPA_ACTOR_SOFT")
@Data
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
@SQLDelete(sql="UPDATE JPA_ACTOR_SOFT SET STATUS='INACTIVE' WHERE AID=? AND UPDATE_COUNT=?") //For soft deletion
@SQLRestriction(" STATUS <> 'INACTIVE' ") // For making inactive records not participating in persistence operations
public class Actor {
  @Id
  @SequenceGenerator(name="gen1",sequenceName = "ACTOR_SEQ",initialValue = 1000,allocationSize = 1)
  @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
  private Integer aid;

  @Column(length = 20)
  @NonNull
  private String aname;
  @Column(length = 30)
  @NonNull
  private String addrs;
  @NonNull
  private Double fee;

  @Column(length = 20)
  @NonNull
  private String category;

  //METADATA columns
  @CreationTimestamp
  @Column(updatable = false)
  private LocalDateTime createDate; //Timestamp feature

  @UpdateTimestamp
  @Column(insertable = false)
  private LocalDateTime updateDate; //Timestamp feature

  @Version
  private Integer updateCount; //Versioning feature

  @Column(length = 20,updatable = false)
  private String createdBy;
  @Column(length = 20,insertable = false)
  private String updatedBy;
  @Column(length = 20)
  private String status="active";
```

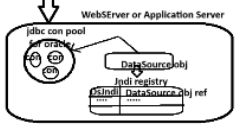
Procedure to cfg Tomcat server managed JDBC con pool in spring Boot MVC App

step1) create JDBC DAtaSource with Jdbc con pool for oracle in Tomcat server by adding <Resource> entries under <Context> tag in Context.xml file of "servers" section from Eclipse Project Explorer.

```
<Resource name="DsJndi" auth="Container"
          type="javax.sql.DataSource" driverClassName="oracle.jdbc.OracleDriver"
          url="jdbc:oracle:thin:@localhost:1521:xe"
          username="system" password="manager" maxTotal="20" maxIdle="10"
          maxWaitMillis="-1"/>
```

(jndi name; wait for con obj until it comes, do not throw exception)

The DataSource object that represents server managed jdbc con pool will be placed in Jndi Registry having the jndi name as the nick name.. we can acess dataSource obj from Jndi registry using this jdni name



Collect reference code from "docs" example web application of < Tomcat_home>\webapps folder (open index.html file and go to JDBC DataSources section)

step2) Add the following entries application.properties to work with server managed jdbc con pool

In applicaiton.properties

```
spring.datasource.jndi-name=java:/comp/env/DsJndi
```

(Jndi name given in <Resource> tag; fixed prefix in tomcat jndi registry)

or

note:: u can comment or ignore remove hikaricp related jdbc properties in application.properties file

note:: server managed jdbc con pool is poosible only while working extenal servers.. not with spring boot supplied embedded Servers.

note:: When we cfg Tomcat server with eclipse IDE , then the Eclipse will not use the external tomcat server , it will gets its own copy of Tomcat server in the workspace folder

Q) In spring boot based web applications (spring boot mvc apps) and spring boot based Restful apps (spring boot Rest Apps) which jdbc con pool is recomanded ?

a) Underlying Server managed jdbc con pool (Not portable)
(This is not recomanded becoz It has to be configured in every external server of every machine)
(works only in exeternal server)

b) Spring boot Supplied standard standalone JDBC con pool (hikaricp, apache dbcp2, tomcat cp, oracle ucp ,....)
(This is recomanded becoz the jdbc con pool will be moved to different machines and servers along with App becoz its part of spring boot setup) (default) (Portable)
(Works in both Embedded and external servers)

JDBC con pools

(good to use)

1.standalone jdbc con pool

These jar oriented

HikariCP (best)
apache dbcp2,
c3P0 and etc..

(Porable across the multiple servers where the app is being deployed and executed)

=>Can be used in both embedded servers and external server

=> supports WODA(Write Once Deploy Any Where) principle

=> These are industry standard JDBC con pools in both standalone apps , web applications & Distributed Apps
(HikariCP is best)

2. server managed jdbc con pool

=>Tomcat managed jdbc con pool
=> weblogic managed jdbc con pool
=> wildfly managed jdbc con pool

These are server managed

(Not portable across the multiple servers)

Can be used only in external server

=> Does not Support WODA becoz in each server the server managed jdbc con pool configurations are different

=>These are not industry standard JDBC Con Pools becoz the con pool configurations will change server to server

=>We can arrange Jndi registries as separate softwares or we can use the Embedded Jndi registries of Servers
eg1:: Cos registry , Rmi registry , DNS registry and etc.. independent jndi registries
eg2:: Tomcat managed jndi registry , Glassfish managed jndi registry and etc.. are server managed jndi registries

---

Debeggguging our Project in Eclipse IDE

=> It is the process knowing the application's flow of execution from the place where breakpoints are applied

=>Breakpoint --->the place in code execution from code flow can be controlled
=> Debugging operations
F5 --> step into
F6 --> Step over
F7 ---> step return
F8 ---> Go to next break point

step1) keep break point in the first line of every method in @Controller class
(note:: make sure that first line java code , not the comment)

step2) Run the Application using debug on server option

```
30  @GetMapping("/report")  // G---- Get
31  public String generateReport(Map<String,Object> map) {
32    System.out.println("EmployeeOperationsController.generateReport()
33    //use service
34    Iterable<Employee> it=empService.showAllEmployees();
35    //keep the result in model attribute (Shared memory)
36    map.put("empList", it);
37    //return LVN
38    return "show_report";
39  }
```

(ctrl+shift+t) double click in the blue line

Soft Deletion in spring MVC and spring data JPA

==========

=> Deleting record perminently from db table is called hard deletion

eg: Ticket cancellation, Booking cancellation and etc..

==========

=> marking the record as DELETED record by changing the status of the record and making record that not participating in CURD operations is called SoftDeletion

eg: BankAccount closing, Employee resignation and etc..

hibernate

@

=> we have Annotations called SQLXXX annotation which allows to configure custom SQL Queries for standard SQL opeartions like insert,update, delete. (@SQLInsert, @SQLDelete, @SQL Update) @SQLDelete annotation is useful to cfg Custom Query for the standard delete operation. This

is veery of UPDATE SQL query of soft deletion to mark the record as the DELETED record for standard delete operation.

Delete operation in Projects

Hard Deletion

(Record deleted physically for ever)

note:Using @SQLXXX annotations we can change the Standard SQL queries of Repositiry methods to our choice custom SQL Queries @Where annotation can be used to specify implicit condtion that should be applied on every query that

executes..

Steps for soft Deletion

=======

@SQLXXX annotations are hibernate annotations

(@Where)

Soft Deletion

(Record is marked for

deletion, but physically it is not deleted)

This annotation helps to make the softly deleted records not participating in any kind of CURD Operations

@Where is removed in the latest versions of hibernate So use @SqlRestriction asthe alternate

the

a) create db table having status column with "active" value for all records..

Using SQL browser of SQL developer

15015 vyr

Columns: Q name

=> create table BOOT_EMP as select empno,ename,job,sal,deptno from emp; ** ADD STATUS colum to db table BOOT_EMP

PK Name

EMPNO

Data Type NUMBER

Size

Not Null

Default

Comm

ENAME

VARCHAR2

4 10

JOB

VARCHAR2

9

=> UPDATE BOOT_EMP SET STATUS='active'

b) Add @SQLDELETE Annotation specifying update query of softdeletion

@SQLDelete(sql="UPDATE BOOT_EMP SET STATUS='INACTIVE' WHERE EMPNO=?") (write on the top of Entity class)

//Commit the records

commit; (to commit the copied records)

SAL

NUMBER

7

DEPTNO STATUS

NUMBER

2

VARCHAR 2

15

BOOT_EMP

Columns Data Model | Constraints Grants Statistics Triggers | Flashback |Depen

Sort.. Filter:

EMPNOENAME JOB

1

7369 SMITH

CLERK

SAL DEPTNO, 1800

STATUS

20 ACTIVE

2

7499 ALLEN

SALESMAN 1600

30 ACTIVE

3

7521 WARD

SALESMAN 1250

30 ACTIVE

4

7566 JONES

MANAGER 2975

20 ACTIVE

c) add @Where annotation specify condition to eleminate softely deleted records from regular CURD Operations. (avoid)

5

6

7

7782 CLARK

8

7788 SCOTT

7654 MARTIN SALESMAN 90000 7698 BLAKE MANAGER 2850 MANAGER 2450 ANALYST 3000

30 ACTIVE

30 ACTIVE

10 ACTIVE

20 ACTIVE

9

7839 KING

PRESIDENT 5000

10 ACTIVE

10

7844 TURNER SALESMAN 3400

30 ACTIVE

11

5 karan

developer 90000

10 ACTIVE

12

7 karan

developer 90000

10 ACTIVE

13

10 mohit

CLERK

14

11 mukesh CLERK

8000 80000

10 ACTIVE

10 ACTIVE

Entity class

==========

```
package com.nt.model;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator; import javax.persistence.Table;
import org.hibernate.annotations.SQLDelete; import org.hibernate.annotations.Where;
import lombok.Data;
```

=> JPA is the software specification giving rules and guidelines to create set of ORM frameworks like Hibernate, IBatis, Eclipse Link and etc..

note:: original repo.delete(-), deleteXxx() methods generates delete SQL Queries for deleting the records.. By using @SQLUpdate we can change those standard SQL Queries with our choice queries using which we can perform soft deletion activity

```
@Entity
@Table(name="boot_emp")
@Data
@SQLDelete(sql = "UPDATE BOOT_EMP SET STATUS='deleted' WHERE EMPNO=?") @Where(clause =
"STATUS <> 'deleted' ")
public class Employee {
}
```

@Id

hibernate

annoations

note:: @Where annotation is deprecated from hibernate 6.2 version (spring boot 3.2),So use @SqlRestriction as the alternate @SqlRestriction("STATUS <> 'deleted')

```
@SequenceGenerator(name = "gen1",sequenceName = "emp_no_seq1",initialValue =3000, allocationSize =
```

1) @GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)

private Integer empno;

private String ename;

private String job;

private Float sal;

private Integer deptno=10; private String status="active";

=>Using Repository methods like delete(-) we can do only soft deletion on this db table using this Entity class.. =>To perform hard deletion on this db table we can"&Query methods of repository eithe with JPQL or Native SQL

5 7654 MARTIN SALESMAN 90000 6 7698 BLAKE MANAGER 2850

30 ACTIVE

30 ACTIVE

7

7782 CLARK MANAGER 2450

10 ACTIVE

8 7788 SCOTT ANALYST

3000

20 ACTIVE

9

7839 KING PRESIDENT 5000

10 ACTIVE

10

7844 TURNER SALESMAN

3400

11

5 karan

developer 90000

12

7 karan

Procedure to cfg Tomcat server managed JDBC con pool in spring Boot MVC App

13

10 mohit CLERK

14

11 mukesh CLERK

developer 90000 8000 80000

30 ACTIVE 10 INACTIVE 10 INACTIVE 10 ACTIVE 10 ACTIVE

d) Run the Application..

Not equal to

softly deleted records

if Entity class of spring data jpa is having @Version property then we need to

provide that version related condition in the @SQLDelete annotation that

is related to soft deletion activitity

```
@Entity
@Data
@Table(name="JPA_ACTOR_SOFT")
@NoArgsConstructor
@AllArgsConstructor
@RequiredArgsConstructor
@SQLDelete(sql="UPDATE JPA_ACTOR_SOFT SET STATUS="INACTIVE' WHERE AID=? AND UPDATE_COUNT=?") //For so
deletion
@SQLRestriction(" STATUS <> 'INACTIVE" ") // For making inactive records not participating in persistence operations public class Actor {
@Id
@SequenceGenerator(name="gen1",sequenceName = "ACTOR_SEQ",initialValue = 1000, allocationSize = 1)
@GeneratedValue(generator = "gen1",strategy = GenerationType.SEQUENCE)
private Integer aid;
@Column(length = 20)
@NonNull
private String aname; @Column(length = 30) @NonNull
private String addrs; @NonNull
private Double fee;
@Column(length = 20) @NonNull
private String category;
//METADATA columns
@CreationTimestamp
@Column(updatable = false)
private LocalDateTime createDate; //timestamp feature
@UpdateTimestamp
@Column(insertable = false)
private LocalDateTime updateDate; //timestamp feature
@Version
private Integer updateCount; //Versioning feature
@Column(length = 20,updatable = false)
private String createdBy;
@Column(length = 20,insertable = false)
```

private String updatedBy; @Column(length = 20)

private String _status="active";

The DataSource object that represents server managed jdbc cor pool will be placed in Jndi Registry having the jndi name as the nick name.. we can acess dataSource obj from Jndi registry using this jdni name

==========

tag

step1) create JDBC DAtaSource with Jdbc con pool for oracle in Tomcat server by adding <Resource> entries under <Context> tag in Context.xml file of "servers" section from Eclipse Project Explorer.

jndi name

<Resource name="DsJndi" auth="Container"

type="javax.sql.DataSource" driverClassName="oracle.jdbc.OracleDriver"

url="jdbc:oracle:thin:@localhost:1521:xe"

username="system" password="manager" maxTotal="20" maxIdle="10" maxWaitMillis="-1" />

wait for con obj until it comes, do not throw exception

step2) Add the following entries application.properties to work with server managed jdbc con

WebServer or Application Server

Collect reference code from "docs" example web application of

jdbc con pool

for oracle/

< Tomcat_home>\webapps folder (open index.html file and go to JDBC DataSources section)

JDBC con pools

DataSourceJobj

Jndi registry

DsJndi DataSource abj ref

pool

In applicaiton.properties

Jndi name given in <Resource> tag

spring.datasource.jndi-name=java:/comp/env/DsJndi

fixed prefix in tomcat jndi registry

or

note:: u can comment or ignore remove hikaricp related jdbc properties in application.properties file

with

note:: server managed jdbc con pool is poosible only while working extenal servers.. not with spring boot supplied embedded Servers.

note:: When we cfg Tomcat server with eclipse IDE, then the Eclipse will not use the

external tomcat server, it will gets its own copy of Tomcat server in the workspace folder

Q) In spring boot based web applications (spring boot mvc apps) and spring boot based Restful apps (spring boot Rest Apps) which jdbc con pool is recomanded?

a) Underlying Server managed jdbc con pool (Not portable)

(This is not recomanded becoz it has to be configured in every external server of every machine) (works only in exeternal server)

b) Spring boot Supplied standard standalone JDBC con pool (hikaricp, apache dbcp2, tomcat cp, oracle ucp,....) (This is recomanded becoz the jdbc con pool

(default)

will be moved to different machines and servers along with App becoz its part of spring boot setup) (Works in both Embedded and external servers)

(Portable)

These

(good to use)

1.standalone jdbc con pool HikariCP (best)

jar oriented apache dbcp2,

c3PO and etc..

(Porable across the multiple servers where the app is being deployed and executed)

=> Can be used in both embedded servers and external server

=> supports WODA (Write Once Deploy Any Where) principle

=> These are industry standard JDBC con pools in both standalone apps,

web applications

& Distributed Apps (HikariCP is best)

2. server managed jdbc con pool

=>Tomcat managed jdbc con pool

=> weblogic managed jdbc con pool =>wlidfly managed jdbc con pool

(Not portable across the multiple servers)

Can be used only in external server

=> Does not Support WODA becoz in each server the server managed jdbc con pool configurations are different

These are server managed

=>These are not industry standard JDBC Con Pools becoz the con pool configurations will change server to server

=>We can arrange Jndi registries as separate softwares or we can use the Embedded Jndi registries of Servers eg1:: Cos registry, Rmi registry, DNS registry and etc.. independent jndi registries

eg2:: Tomcat managed jndi registry, Glassfish managed jndi registry and etc.. are server managed jndi registries

Debegguging our Project in Eclipse IDE

==========

=> It is the process knowing the application's flow of execution from the place where breakpoints are applied

=>Breakpoint --->the place in code execution from code flow can be controlled => Debugging operations

F5 --> step into

F6 --> Step over

F7 ---> step return

F8 ---> Go to next break point

30

31

(ctrl+shift+b)

step1) keep break point in the first line of every method in @Controller class (note:: make sure that first line java code, not the comment)

step2)

Run the Application using debug on server option

double 32

click in

the blue line 33

@GetMapping("/report") // G---- Get

public String generateReport(Map<String,Object> map) {
System.out.println("EmployeeOperationsController.generateReport()'

Iterable<Employee> it=empService.showAllEmployees(); //keep the result in model attribute (Shared memory)
map.put("empList", it);

//use service

34

35

36

37

39

38 | }

//return LVN

return "show_report";

step3) Give request from browser and accept switching to debug mode use f5, f6, f7 keys as required

recomandations: f5 --> for user-defined method call f6--> for prefined method call

f7---->To go to end of the method definitation

from the middle of the definitation

note:: In any method definiation execution, if u want to execute the method from the beginning by dropping from the middle, we need to use drop to frame option