Date : 11/02/2021
Spring Boot 9AM
Mr. RAGHU

Spring AOP:-
https://www.youtube.com/c/NareshIT/search?query=AOP%20raghu

Redis As DB and Cache:-
https://www.youtube.com/c/NareshIT/search?query=redis%20cache%20raghu

Working with Multiple Databases:
https://www.youtube.com/watch?v=nzszxQbQ5WU&list=PLVlQHNRLflP9XSWeY4x4FLwnL3UOIxnTr&index=20

Spring Data JPA: findBy

*) Projections:
   By default findBy is fetching all columns data. (List<T>)
   But now, we are using Projections that helps us to get
   only specific columns (List<CustomType>)

S#1 Define one interface inside Repository (Inner Interface)
    that contains variables equal get method which need to be project.

```
    interface  <CustomType> {
      DataType  get<VariabeName>();
    }
```

S#2 Use This interface as ReturnType to findBy method

```
    List<CustomType>  findBy<VariablesConditions>(<Params>);
```

S#3 Call above method inside Runner/Service...etc For testing.

--Ex---
1. Model

```
class Student {
   .....;
   Integer sid;
   String sname;
   Double sfee;
}
```

2.

```
interface StudentRepository extends JpaRepository<Student,Integer> {

  interface MyView {
     String getSname();
     Double getSfee();
  }

  // SQL: select sname,sfee from student sid=?
  List<MyView>  findBySid(Integer sid);
```

```
}

3. Runner class
  @Autowired
  StudentRepository repo;

 List<MyView> list = repo.findBySid(10);
 //for each..print

==================Full code=============================
1. Model
package in.nareshit.raghu.model;

import javax.persistence.Entity;
import javax.persistence.Id;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

@Data
@NoArgsConstructor
@AllArgsConstructor
@Entity
public class Student {
        @Id
        private Integer stdId;
        private String stdName;
        private Double stdFee;
        private String stdCourse;

}

2. Repository Interface
package in.nareshit.raghu.repo;

import java.util.List;

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

import in.nareshit.raghu.model.Student;

public interface StudentRepository
        extends JpaRepository<Student, Integer> {

        //1. define one view interface
        interface MyView {
                 String getStdName();
                 Double getStdFee();
        }

        //SQL: select sname,sfee from student sid<=?
        List<MyView> findByStdIdLessThanEqual(Integer sid);

        //2. define one view interface
        interface MyViewTwo {
                String getStdName();
```

```
		String getStdCourse();
	}

	//SQL: select sname, course from student where sname is not
null
	List<MyViewTwo> findByStdNameIsNotNull();

}
```

3. Runner for Insert

```
package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;

import in.nareshit.raghu.model.Student;
import in.nareshit.raghu.repo.StudentRepository;

//@Component
public class StudentDataInsertRunner implements CommandLineRunner {
	@Autowired
	private StudentRepository repo;

	public void run(String... args) throws Exception {
		repo.save(new Student(101, "A", 300.0, "JAVA"));
		repo.save(new Student(102, "N", 400.0, "UI"));
		repo.save(new Student(103, "B", 500.0, "JAVA"));
		repo.save(new Student(104, "G", 800.0, "AWS"));
	}

}
```

4. Runner for DataFetch

```
package in.nareshit.raghu.runner;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

import in.nareshit.raghu.repo.StudentRepository;

@Component
public class DataFetchRunner implements CommandLineRunner {
	@Autowired
	private StudentRepository repo;

	public void run(String... args) throws Exception {
		/*
		List<MyView> list =
repo.findByStdIdLessThanEqual(200);

		for(MyView m : list ) {
			System.out.println(m.getStdName() + " -" +
m.getStdFee() );
		}*/

		repo.findByStdNameIsNotNull()
```

```
                .stream()
                .map(ob->ob.getStdName()+"-"+ob.getStdCourse())
                .forEach(System.out::println);
                ;
        }

}
```

5. application.properties

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

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

==========

@Query (non-select operations too, Complex Queries)/ findBy

---

*) Transactions: This is Database concept.

Two operations:

- commit()
- rollback()

Q) When Tx are required?

A) For Every non-select operation Tx are required.

save(obj)--INSERT--

update(obj) -- UPDATE

DELETE(obj) -- DELETE

*) Tx are not required for select operations.

*) Here we have two functions/operations/commands

a. commit : It will update data from buffer to Database (save it)

b. rollback : it will cancel operation/data from buffer
do not effect to Database.

*) Buffers are part of RAM only.

----------MySQL Commands----------

```
mysql> create table student(sid int,sname varchar(20),
         sfee double, scourse varchar(20));

mysql> set autocommit =0;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from student;
Empty set (0.00 sec)

mysql> insert into student values(10,'a',330,'JAVA');
Query OK, 1 row affected (0.02 sec)

mysql> select * from student;
+--------+------------+---------+----------+
| std_id | std_course | std_fee | std_name |
+--------+------------+---------+----------+
```

```
|     10 | a           |     330 | JAVA     |
+--------+-------------+---------+----------+
1 row in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.13 sec)

mysql> select * from student;
Empty set (0.00 sec)
```

```
mysql> insert into student values(10,'a',330,'JAVA');
Query OK, 1 row affected (0.02 sec)

mysql> insert into student values(11,'B',430,'JAVA');
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.07 sec)

mysql> select * from student;
+--------+-------------+---------+----------+
| std_id | std_course | std_fee | std_name |
+--------+-------------+---------+----------+
|     10 | a           |     330 | JAVA     |
|     11 | B           |     430 | JAVA     |
+--------+-------------+---------+----------+
2 rows in set (0.00 sec)

mysql> insert into student values(12,'C',530,'JAVA');
Query OK, 1 row affected (0.00 sec)

mysql> select * from student;
+--------+-------------+---------+----------+
| std_id | std_course | std_fee | std_name |
+--------+-------------+---------+----------+
|     10 | a           |     330 | JAVA     |
|     11 | B           |     430 | JAVA     |
|     12 | C           |     530 | JAVA     |
+--------+-------------+---------+----------+
3 rows in set (0.00 sec)

mysql> rollback;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from student;
+--------+-------------+---------+----------+
| std_id | std_course | std_fee | std_name |
+--------+-------------+---------+----------+
|     10 | a           |     330 | JAVA     |
|     11 | B           |     430 | JAVA     |
+--------+-------------+---------+----------+
2 rows in set (0.00 sec)
```

Q) Where Tx is useful?
A) Connected Queries Tx is must.

Account

```
  ----------------------------
    accId   accName   bal
  ----------------------------
     100      A       500.0         - 200 SQL#1
     101      B       300.0         + 200 SQL#2
  ----------------------------

Transfer Money from A to B 200.0 Rs/-

SQL#1
  UPDATE Account SET bal=bal-200 WHERE accId=100;

  bal--> 101# 300

SQL#2
  UPDATE Account SET bal=bal+200 WHERE accId=101;

  bal--> 100# 500

> commit/rollback
-----------------------------------
execute SQL#1 && SQL#2
if
 both success: commit
else
 any one fail: rollback
-----------------------------------
mysql> create table account(aid int,aname varchar(10),bal double);
Query OK, 0 rows affected (0.84 sec)

mysql> insert into account values(101,'A',1000.0);
Query OK, 1 row affected (0.03 sec)

mysql> insert into account values(102,'B',2000.0);
Query OK, 1 row affected (0.00 sec)

mysql> commit;
Query OK, 0 rows affected (0.05 sec)

mysql> select * from account;
+------+-------+------+
| aid  | aname | bal  |
+------+-------+------+
|  101 | A     | 1000 |
|  102 | B     | 2000 |
+------+-------+------+
2 rows in set (0.00 sec)

mysql> update account set bal=bal-200.0 where aid=101;
Query OK, 1 row affected (0.04 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set bal=bal+200.0 where aid=105;
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> rollback
```

```
    -> ;
Query OK, 0 rows affected (0.06 sec)

mysql> select * from account;
+------+-------+------+
| aid  | aname | bal  |
+------+-------+------+
|  101 | A     | 1000 |
|  102 | B     | 2000 |
+------+-------+------+
2 rows in set (0.00 sec)

mysql> update account set bal=bal-200.0 where aid=101;
Query OK, 1 row affected (0.02 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> update account set bal=bal+200.0 where aid=102;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> commit;
Query OK, 0 rows affected (0.04 sec)

mysql> select * from account;
+------+-------+------+
| aid  | aname | bal  |
+------+-------+------+
|  101 | A     |  800 |
|  102 | B     | 2200 |
+------+-------+------+
2 rows in set (0.00 sec)

mysql>
---JDBC-----------------------------------------------
try {
 Connection con= DriverManager.getConnection(..);
 con.setAutoCommit(false);

  SQL#1 (Prepared Stmt)
  SQL#2 (Prepared Stmt)

  con.commit(); //inform db to save results in in tables from buffer
} catch(SQLException exp) {
  con.rollback(); // cancel data from buffer;
}

---Hibernate-----------------------
Transaction tx = null;
try {
  tx = session.beginTx();
   //operations..
  tx.commit();
} catch(Exception ex) {
  tx.rollback();
}
-------------------------------------
Spring F/w --
```

```
@Transactional
void m1() {
  ......
}
Spring AOP
afterReturningAdvice() //success
{
 tx.commit()
}
afterThrowingAdvice() //fail
{
 tx.rollback();
}
-----------------------------------
Spring Boot
--DO NOT WRITE CODE---

PlatformTransactionManager(I)
  IS-A
JpaTransactionManager(C)

EntityManagerFactory(I)
  IS-A
SessionFactory(I)
  IS-A
SessionFactoryDelegatingImpl(C)
```