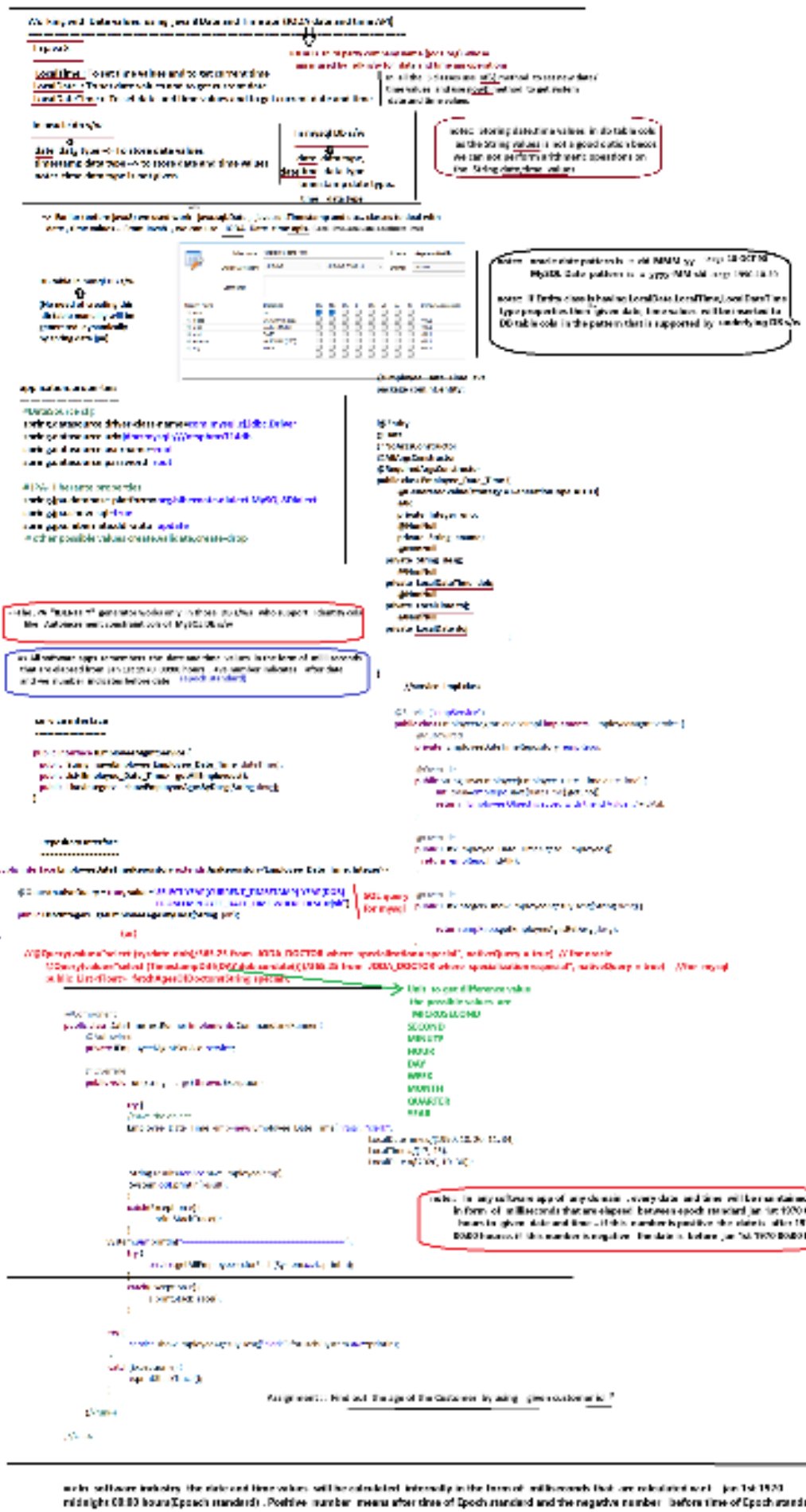

Working with Date values using java 8 Date and Time api (JODA date and time API)

In java 8

JODA is third party company name (joda.org) whose api is used by jdk s/w for date and time api operations

LocalTime: To set time values and to get current time LocalDate : To set date values and to get current date LocalDateTime :: To set date and time values and to get current date and time

In oracle db s/w

↓

date data type --> To store date values

timestamp data type --> to store date and time values

note: time data type is not given

with

In mysql Db s/w

In all the 3 classes use of(-) method to set new date/ time values and use now() method to get system date and time values

date data type, datetime data type timestamp date type.. time data type

=> Earlier (before java8) we used work java.sql.Date, java.sql.Timestamp and etc.. classes to deal with date, time values.. From Java8, we can use JODA Date -time apis. (LocalTime, LocalDate, LocalDateTime)

note:: Storing date, time values in db table cols as the String values is not a good option becoz we can not perform arithmetic opeations on the String date,time values

DB table in mysql DB s/w

↑

(No need of creating this db table manullly, will be generated dynamically by spring data jpa)

Table Name: employee_date_time Charset/Collation: utf8mb4

utf8mb4_0900_ai_ci

Schema: ntspbms616db Engine: InnoDB

Comments:

Column Name

eno

desg

dob

doj

ename

toj

Datatype INT

VARCHAR(255) DATETIME(6) DATE

VARCHAR(255)

TIME

UN ZF AI G

nn

0 OOOO

oooo

0000000 -

U

U

Default/Expression

NULL

NULL

NULL

NULL

U

NULL

U

S0000000

PK NN

B

U

0000000

0000000

note:: oracle date pattern is :: dd-MMM-yy ->eg: 10-OCT-90 MySQL Date pattern is: yyyy-MM-dd ->eg: 1990-10-20

the

note:: if Entity class is having LocalDate, LocalTime, LocalDateTime type properties then given date, time values will be inserted to DB table cols in the pattern that is supported by underlying DB s/w

application.properties

==========

#DataSource cfg

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql:///ntspbms714db

spring.datasource.username=root spring.datasource.password=root

#JPA-Hiberante properties

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto-update

 # other possible values create,validate,create-drop

=>The JPA "IDENTITY" generator works only in those DB s/ws who support Identity cols like Autoincrement

constraint cols of MySQL DB s/w

=> All software apps remembers the date and time values in the form of milli seconds that are elapsed from jan 1st 1970 00:00 hours. +ve number indicates after date and -ve number indicates before date (epoch standard)

service interface

public interface IEmployee MgmtService {

public String saveEmployee (Employee_Date_Time dateTime); public List<Employee_Date_Time> getAllEmployees();

public List<Integer> showEmployeeAgesByDesg(String desg);

}

//Employee_Date_Time.java

package com.nt.entity;

i@Entity

@Data

@NoArgsConstructor

@AllArgsConstructor

@RequiredArgsConstructor

public class Employee_Date_Time {

@GeneratedValue(strategy = GenerationType.AUTO)

@Id

private Integer eno;

@NonNull

private String ename;

@NonNull

private String desg;

@NonNull

private LocalDateTime dob;

@NonNull

private LocalTime toj; @NonNull

private LocalDate doj;

//service Impl class

@Service("empService")

public class EmployeeMgmtServiceImpl implements IEmployeeMgmtService { @Autowired

private EmployeeDateTimeRepository empRepo;

@Override

public String saveEmployee(Employee_Date_Time dateTime) {

int idVal-empRepo.save(dateTime).getEno();

return "Employee Object is saved with the id Value ::"+idVal;

}

Repository Interface

==========

public interface Employee DateTimeRepository extends JpaRepository<Employee_Date_Time, Integer> {

@Override

public List<Employee_Date_Time> getAllEmployees() { return empRepo.findAll();

}

@Query(nativeQuery = true, value="SELECT YEAR(CURRENT_TIMESTAMP)-YEAR(DOB) FROM EMPLOYEE_DATE_TIME WHERE DESG=:job") public List<Integer> getEmployeeAgesByDesg(String job);

SQL query @Override

for mysql public List<Integer> show EmployeeAgesByDesg(String desg) {

}

(or)

return empRepo.getEmployeeAgesByDesg(desg);

}

//@Query(value="select (sysdate-dob)/365.25 from JODA_DOCTOR where specialization=:special", nativeQuery = true) // for oracle @Query(value="select (Timestamp Diff(DAY.dob,curdate()))/365.25 from JODA_DOCTOR where specialization=:special", nativeQuery = true) //for mysql public List<Float> fetchAgesOfDoctors(String special),

Unit to get difference value

@Component

public class DateTimeTestRunner implements CommandLineRunner { @Autowired

private IEmployeeMgmtService service;

the possible values are

MICROSECOND

SECOND

MINUTE

HOUR

@Override

public void run(String... args) throws Exception {

try {

//save the object

DAY

WEEK

MONTH

QUARTER YEAR

Employee_Date_Time emp-new Employee_Date_Time("raja", "clerk",

LocalDateTime.of(1990, 10, 20, 11, 34), LocalTime.of(17,45),

```
LocalDate.of(2020, 10, 30));
String result-service.saveEmployee(emp);
System.out.println(result);
}
catch(Exception e) {
e.printStackTrace();
}
System.out.println("++++++++++++++++++++++++++++++++++++++");
try {
service.getAllEmployees().forEach(System.out::println);
catch(Exception e) {
e.printStackTrace();
}
try {
service.showEmployeeAgesByDesg("clerk").forEach(System.out::println);
}
catch(Exception e) { e.printStackTrace();
}
}//run(-)
}//class
```

note:: In any software app of any domain, every date and time will be maintained in form of milliseconds that are elapsed between epoch standard jan 1st 1970 00:00 hours to given date and time.. if this number is positive the date is after 1970 jan 1st 00:00 hourse. if this number is negative the date is before jan 1st 1970 00:00 hours

Assignment :: Find out the age of the Customer by using given customer id ?

=>In software industry the date and time values will be calculated internally in the form of milliseconds that are calculated w.r.t jan 1st 1970 midnight 00:00 hours (Epoach standard). Positive number means after time of Epoch standard and the negative number before time of Epoch standard

Age Calucator logics w.r.t Oracle DB s/w using NAtive SQL Query based

=========

Code in Repository Interface

@Query method

@Query(value="select (sysdate-dob)/365.25 from JPA_JODA_DATE_TIME WHERE PID=:id", nativeQuery = true) public float calculateAgeByPid(int id);

Code in Service Interface

public float getAgeByPid(int id);

Code in Service Impl class

@Override

public float getAgeByPid(int id) {

```
return personRepo.calculateAgeByPid(id);
}
```

Code in Runner class

```
try {
System.out.println("Person Age is ::"+personService.getAgeByPid(1));
}
catch (Exception e) {
}
e.printStackTrace();
```