Date : 22/04/2021
Spring Boot 9AM
Mr. RAGHU

Working with JPA Writer: CsvToMySQLUsingJPA

a. Add Spring Data JPA Dependency

```
<dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

b. Provide JPA Keys in proeprties

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

c. Provide JPA Annotation over model class

```
 @Entity, @Id ..etc
```

d. Use JpaItemWriter and create bean

```
 @Autowired
 private EntityManagerFactory emf;

 @Bean
 public ItemWriter<Product> writer() {
    JpaItemWriter<Product> writer = new JpaItemWriter<>();
    writer.setEntityManagerFactory(emf);
    return writer;
 }
```

=======Full code================

Name : springBoot2CsvToMySQLUsingJpa
Dep : batch , lombok, MySQL, Data JPA

1. Model class

```
package in.nareshit.raghu.model;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;

import lombok.Data;

@Data
@Entity
@Table(name="prodtab")
public class Product {
        @Id
        @Column(name="pid")
        private Integer prodId;
        @Column(name="pcode")
        private String prodCode;
        @Column(name="pcost")
```

```
        private Double prodCost;
        @Column(name="pgst")
        private Double prodGst;
        @Column(name="pdiscount")
        private Double prodDiscount;
}
```

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

spring.batch.job.enabled=false
spring.batch.initialize-schema=always
```

3. Processor

```
package in.nareshit.raghu.processor;

import org.springframework.batch.item.ItemProcessor;

import in.nareshit.raghu.model.Product;

public class ProductProcessor
        implements ItemProcessor<Product, Product>
{

        public Product process(Product item) throws Exception {
                //JDK 10 # local variable type inference
                // [best datatype is decided at compiletime]
                var cost = item.getProdCost();

                //calculations
                var gst = cost * 12/100.0;
                var disc = cost * 8/100.0;

                //set data back to actual object
                item.setProdGst(gst);
                item.setProdDiscount(disc);

                return item;
        }
}
```

4. Listener

```
package in.nareshit.raghu.listener;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;

public class MyJobListener
```

```
        implements JobExecutionListener
{

        private static final Logger LOG =
LoggerFactory.getLogger(MyJobListener.class);

        public void beforeJob(JobExecution je) {
                LOG.info("BEFORE STARTING JOB {},
{}",je.getStatus(),je.getStartTime());
        }

        public void afterJob(JobExecution je) {
                LOG.info("AFTER FINISHING JOB {},
{}",je.getStatus(),je.getEndTime());

        }
}
```

5. BatchConfig

```
package in.nareshit.raghu.config;

import javax.persistence.EntityManagerFactory;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import
org.springframework.batch.core.configuration.annotation.EnableBatchPro
cessing;
import
org.springframework.batch.core.configuration.annotation.JobBuilderFact
ory;
import
org.springframework.batch.core.configuration.annotation.StepBuilderFac
tory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.database.JpaItemWriter;
import org.springframework.batch.item.file.FlatFileItemReader;
import
org.springframework.batch.item.file.mapping.BeanWrapperFieldSetMapper;
import org.springframework.batch.item.file.mapping.DefaultLineMapper;
import
org.springframework.batch.item.file.transform.DelimitedLineTokenizer;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.ClassPathResource;

import in.nareshit.raghu.listener.MyJobListener;
import in.nareshit.raghu.model.Product;
import in.nareshit.raghu.processor.ProductProcessor;

@EnableBatchProcessing
@Configuration
```

```
public class BatchConfig {

	//1. reader object
	@Bean
	public ItemReader<Product> reader() {
		//JDK 1.7 Collections Type Inference
		FlatFileItemReader<Product> reader = new
FlatFileItemReader<>();
		reader.setResource(new
ClassPathResource("products.csv"));
		reader.setLineMapper(new DefaultLineMapper<>() {{
			setLineTokenizer(new DelimitedLineTokenizer()
{{
				setDelimiter(DELIMITER_COMMA);

setNames("prodId","prodCode","prodCost");
			}});
			setFieldSetMapper(new
BeanWrapperFieldSetMapper<>() {{
				setTargetType(Product.class);
			}});
		}});

		return reader;
	}
	//2. processor object
	@Bean
	public ItemProcessor<Product,Product> processor(){
		return new ProductProcessor();
	}

	/*@Autowired
	private DataSource dataSource;
	*/
	@Autowired
	private EntityManagerFactory emf;

	//3. writer object
	@Bean
	public ItemWriter<Product> writer(){
		JpaItemWriter<Product> writer = new JpaItemWriter<>();
		writer.setEntityManagerFactory(emf);
		/*JdbcBatchItemWriter<Product> writer = new
JdbcBatchItemWriter<>();
		writer.setDataSource(dataSource);
		writer.setSql("INSERT INTO
PRODUCTS(PID,PNAME,PAMT,PGST, PDISC)
VALUES(:prodId,:prodCode,:prodCost,:prodGst,:prodDiscount)");
		writer.setItemSqlParameterSourceProvider(new
BeanPropertyItemSqlParameterSourceProvider<Product>());
		*/
		return writer;
	}
	//4. listener object
	@Bean
	public JobExecutionListener listener(){
		return new MyJobListener();
```

```
	}
	//5. autowired SBF
	@Autowired
	private StepBuilderFactory sf;
	//6. Step object
	@Bean
	public Step stepA(){
		return sf.get("stepA")//name
				.<Product,Product>chunk(3)//I,O,chunk
				.reader(reader())
				.processor(processor())
				.writer(writer())
				.build()
				;
	}
	//7. autowired JBF
	@Autowired
	private JobBuilderFactory jf;
	//8. Job object
	@Bean
	public Job jobA(){
		return jf.get("jobA")//name
				.listener(listener())
				.incrementer(new RunIdIncrementer())
				.start(stepA())
				.build();
	}
}
```

6. Runner class

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

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobParameters;
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class MyJobRunner implements CommandLineRunner {

	@Autowired
	private JobLauncher launcher;
	@Autowired
	private Job jobA;


	public void run(String... args) throws Exception {
		JobParameters params = new JobParametersBuilder()
				.addLong("time",
System.currentTimeMillis())
				.toJobParameters();

		launcher.run(jobA, params);
	}
```

```
}

7. products.csv
10,PEN,200.0
11,BOOK,500.0
12,BOTTLE,600.0
13,MOBILE,1800.0
14,MOUSE,300.0
15,KEYBRD,900.0
16,BAG,600.0
==========================================================
      JAXB : Java Architecture for XML Binding

Marshalling  :- Converting Java Object to XML Format
Unmarshalling:- Converting XML to Java Object Format

https://docs.spring.io/spring-ws/site/reference/html/oxm.html

SpringBatchMongoDbToXml:-

Name : SpringBoot2MySQLToXML
Dep : Batch API, Lombok, MySQL

a. Spring OXM
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-oxm</artifactId>
</dependency>

JAXB-API:
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
</dependency>

XML.BIND-API
<dependency>
    <groupId>jakarta.xml.bind</groupId>
    <artifactId>jakarta.xml.bind-api</artifactId>
</dependency>

JAXB-RUNTIME
<dependency>
    <groupId>org.glassfish.jaxb</groupId>
    <artifactId>jaxb-runtime</artifactId>
</dependency>

============Database setup=================
1. create table

>  drop database boot9am;
> create database boot9am;
> use boot9am;
 create table usertab(uid int, uname varchar(20),
   urole varchar(20),udept varchar(20));

2. insert data
```

```
insert into usertab values(10,'A','ADMIN','DEV');
insert into usertab values(11,'B','ADMIN','QA');
insert into usertab values(12,'C','SE','DEV');
insert into usertab values(13,'D','TE','QA');
insert into usertab values(14,'E','ADMIN','BA');
insert into usertab values(15,'F','MG','BA');

>commit;
```

---

1. Model class

```
package in.nareshit.raghu.model;

import javax.xml.bind.annotation.XmlRootElement;

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

@Data
@NoArgsConstructor
@AllArgsConstructor
@XmlRootElement(name = "user")
public class User {

        private Integer userId;
        private String userName;
        private String userRole;
        private String userDept;

}
```

2. BatchConfig

```
package in.nareshit.raghu.config;

import javax.sql.DataSource;

import org.springframework.batch.core.Job;
import org.springframework.batch.core.JobExecution;
import org.springframework.batch.core.JobExecutionListener;
import org.springframework.batch.core.Step;
import
org.springframework.batch.core.configuration.annotation.EnableBatchPro
cessing;
import
org.springframework.batch.core.configuration.annotation.JobBuilderFact
ory;
import
org.springframework.batch.core.configuration.annotation.StepBuilderFac
tory;
import org.springframework.batch.core.launch.support.RunIdIncrementer;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.batch.item.ItemReader;
import org.springframework.batch.item.ItemWriter;
import org.springframework.batch.item.database.JdbcCursorItemReader;
import org.springframework.batch.item.xml.StaxEventItemWriter;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
```

```
import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.FileSystemResource;
import org.springframework.oxm.jaxb.Jaxb2Marshaller;

import in.nareshit.raghu.model.User;

@EnableBatchProcessing
@Configuration
public class BatchConfig {

	@Autowired
	private DataSource dataSource;

	@Bean
	public ItemReader<User> reader(){
		JdbcCursorItemReader<User> reader = new
JdbcCursorItemReader<>();
		reader.setDataSource(dataSource);
		reader.setSql("SELECT UID, UNAME, UROLE, UDEPT FROM
USERTAB");
		//reader.setRowMapper(new UserRowMapper());
		reader.setRowMapper(
				(rs,n)->
				new User(
					rs.getInt("uid")
					,rs.getString("uname")

,rs.getString("urole"),

					rs.getString("udept")
					));
		return reader;
	}

	@Bean
	public ItemProcessor<User,User> processor(){
		return item->item;
		//return new UserProcessor();
	}

	@Bean
	public Jaxb2Marshaller marshaller() {
		Jaxb2Marshaller marshaller = new Jaxb2Marshaller();
		marshaller.setClassesToBeBound(User.class);
		return marshaller;
	}

	@Bean
	public ItemWriter<User> writer(){
		// StAX = Streaming API for XML (JAXB)
		StaxEventItemWriter<User> writer = new
StaxEventItemWriter<>();
		//XML file location
		writer.setResource(new
FileSystemResource("E:/myouts/usersdata.xml"));
		writer.setMarshaller(marshaller());
		writer.setRootTagName("users");
		return writer;
```

```
	}
	@Bean
	public JobExecutionListener listener(){
		//return new MyJobListener();
		return new JobExecutionListener() {
			public void beforeJob(JobExecution je) {
				System.out.println(
						"Starting : "
+je.getStatus());
			}
			public void afterJob(JobExecution je) {
				System.out.println(
						"Ending : "
+je.getStatus());
			}
		};
	}

	@Autowired
	private StepBuilderFactory sf;

	@Bean
	public Step stepA(){
		return sf.get("stepA")
				.<User,User>chunk(3)
				.reader(reader())
				.processor(processor())
				.writer(writer())
				.build();
	}
	@Autowired
	private JobBuilderFactory jf;

	@Bean
	public Job jobA(){
		return jf.get("jobA")
				.listener(listener())
				.incrementer(new RunIdIncrementer())
				.start(stepA())
				.build();
	}
}
```

3. properties file

```
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.batch.job.enabled=false
spring.batch.initialize-schema=always
```

4. Runner class

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

import org.springframework.batch.core.Job;
```

```
import org.springframework.batch.core.JobParametersBuilder;
import org.springframework.batch.core.launch.JobLauncher;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.stereotype.Component;

@Component
public class MyJobRunner implements CommandLineRunner {

        @Autowired
        private JobLauncher launcher;
        @Autowired
        private Job jobA;

        public void run(String... args) throws Exception {
                launcher.run(jobA, new JobParametersBuilder()
                                .addLong("time",
System.currentTimeMillis())
                                .toJobParameters());
                System.out.println("DONE");
        }
}
```

*) Check XML File location : E:\myouts\usersdata.xml

==========

Note:

CSV:

FlatFileItemReader

FlatFileItemWriter

JDBC:

JdbcBatchItemWriter

JdbcCursorItemReader

MongoDB:

MongoItemReader

MongoItemWriter

JPA:

JpaPagingItemReader

JpaItemWriter

XML:

StaxEventItemReader

StaxEventItemReader

JSON:

JsonItemReader

JsonFileItemWriter

https://docs.spring.io/spring-batch/docs/current/reference/html/readersAndWriters.html#jsonReadingWriting

==========

JobRepository:

Repository means memory, here tables created for storing details of execute batches.

=> To avoid this tables creation, we can use

```
      H2 Database as Repository and add
   spring.batch.initialize-schema=embedded
   (default value is embedded)

=> Never store these details in Temp Database also.
   spring.batch.initialize-schema=never

=> Use External Database Ex: MySQL,Oracle and create tables
  spring.batch.initialize-schema=always

=> initialize-schema internally following one Enum
    DataSourceInitializationMode {
       ALWAYS, EMBEDDED, NEVER
    }

--Test Queries---
a. FETCH INSTANCE ID USING JOB NAME
SELECT JOB_INSTANCE_ID FROM  batch_job_instance WHERE JOB_NAME='jobA';

b. Ftech Date and Time + Status data using Instance Id

SELECT START_TIME, END_TIME, STATUS  FROM batch_job_execution WHERE
JOB_INSTANCE_ID IN (
   SELECT JOB_INSTANCE_ID FROM  batch_job_instance WHERE
JOB_NAME='jobA'
);

c. Fetch Job Execution Id using Instance Id
SELECT JOB_EXECUTION_ID  FROM batch_job_execution WHERE
JOB_INSTANCE_ID IN (
   SELECT JOB_INSTANCE_ID FROM  batch_job_instance WHERE
JOB_NAME='jobA'
);

d. Fetch Step details using  Job Execution
SELECT STEP_NAME, START_TIME, END_TIME, STATUS  FROM
batch_step_execution WHERE JOB_EXECUTION_ID IN(
  SELECT JOB_EXECUTION_ID  FROM batch_job_execution WHERE
JOB_INSTANCE_ID IN (
      SELECT JOB_INSTANCE_ID FROM  batch_job_instance WHERE
JOB_NAME='jobA'
   )
);
=====================================================
*) Key:  spring.batch.job.enabled default : true
This key indicates Execute jobs on application startup once.
To avoid batch processing on app startup provide value as false

Bcoz, we wrote one Runner class for JobLaucnhing.
So, spring.batch.job.enabled=false

*) in realtime, Batch APIs are executed using schedulers
a. At Starter: @EnableScheduling
b. Runner class as
@Component
public class MyJobRunner
{
```

```
        @Autowired
        private JobLauncher launcher;

        @Autowired
        private Job jobA;

        @Scheduled(cron = "0 0 13 * * *")
        public void execute() throws Exception {
                JobParameters params = new JobParametersBuilder()
                                        .addLong("time",
System.currentTimeMillis())
                                        .toJobParameters();

                launcher.run(jobA, params);
        }

}

*) we can pass value as expression
 @Scheduled(cron = "${my.input.data}")
  (option args)
  --my.input.data=0  0 13 * * *
```