Spring Batch App converting DB table records into CSV file (Excel file)

==============:

=========

======

(Spring Boot 3.x setup)

usecase :=> Sending Bank Account statement to customer from db table in the form of excel sheet/csv file
=> Giving VotersList of PS(poling station) to the Leader as csv file by collecting information from db table =>
University publishing Elected students for different companies

paid

=> Publishing list of students who have not fees

=> publishing list of customers in a societey showing their power bill details

and etc..

=>For reading from Db table we can use JdbcCursorItemReader<T>

and dues

and for writing processed data to csv file/json file/text file we can use FlatFileItemWriter<T>

DB script creating huge number of random records in mysql Db table

Step-1(Create Database)

====== ========

create Database EXAM_DATA; (pr)

Step-2(Use Database)

==================

use EXAM_DATA;

In mysql workbench

launch mysql workbench -->

name ::NTSPBMS617DB1---> apply -->next ---> ....

(pr)

select

NTSPBMS617DB1

Step-3 (Create Table)

===========

=====

CREATE TABLE `EXAM_RESULT

(

`id`

bigint (20) NOT NULL AUTO_INCREMENT,

);

`dob`

`Semester

`percentage`

Table Name: EXAM RESULT Charset/Collation: Default Charset

Schema:

ntspbms715db1

Default Collation

Engine:

InnoDB

Comments:

Column Name

timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP,

id

dob

int(11)

DEFAULT NULL,

◇ semester

float

DEFAULT NULL,

percentage

Datatype INT TIMESTAMP INT(11) FLOAT

O

00000

U U

000008

PK NN UQ B UN

00000

300000

PRIMARY KEY (`id`)

Step-4 (Create Procedure to Insert)

====

DELIMITER $$

====

CREATE PROCEDURE generate_EXAM_RESULT()

BEGIN

DECLARE i INT DEFAULT 0;

00000

AI G Default/Expression

#00000

L

U

00000

CURRENT_TIMESTAMP

NULL

NULL

WHILE i < 500000 DO

INSERT INTO `EXAM_RESULT` (`dob`, `percentage`,`Semester`) VALUES (

FROM_UNIXTIME(UNIX_TIMESTAMP('2000-01-01 01:00:00')+FLOOR(RAND()*31536000)),

ROUND(RAND()*100,2),

SET i = i + 1;

END WHILE;

END$$

DELIMITER;

`1 );`

Step-5 (Call Procedure to Insert 50 Data)=> Takes Least 30-45 Min to Insert the records

====

CALL generate_EXAM_RESULT();

Step-6 (Check the Records)

========

select * from EXAM_RESULT;

Story board of batching (Db table records to csv file)

==

Expand NTSPBMS715db1--->

right click on stored preocedures --> create procedure

change procedure name generate_exam_result

and write this code b/w begin and end blocks

while i<500000 do

insert into exam_result (dob,percentage,semester)

values(from_unixtime(unix_timestamp ('2000-01-01 01:00:00')+floor(rand()*31536000)), round(rand()*100,2),1);

set i=i+1;

end while;

=>double click on procedure name in MySQL workbench then the procedure will be called automatically

|---> Model class name

Exam Result object

Model class obj)

id:

dob:

percentage:

--> apply ---> next ---> next

ChunkSize::1

RowMapper :: To create Model class obj from db table row FiledSetMapper :: For tokens to Model class obj FiledExtractor :: For Model class obj to Tokens

LineMapper: For getting Line from CSV file LineAggregator :: For creating Line in CSV file

FlatFileItemWriter<Exam Result>

1.specify the name and location csv file writer.setResource(". .")

....

mysal Db s/w S EXAM RESULT

(db table)

JdbcCursorItemReader<Exam Result>

1. create DataSource obj and link to reader object reader.setDataSource(-)

2. specify SELECT SQL Query to

get records

rob table

ESTER

reader.setSql("SELECT ID,DOB,PERCENTAGE,SEM FROM EXAM_RESULT")

record

3. Process each record of the generated ResultSet using RowMapper to write each into one

Modelclass object reader.setRowMapper(....)

Exam ResultItemProces Exam Result, ExamResult>.

(filter student details

who are having >=90 percentage)

2.Specify the FieldExtractor to get content from Model object fileds writer.setFieldExtractor(......)

3. Specify LineAggregator to prepare

lines in csv file using the above field values having specified delimeter (",")

Exam Result object

Model class obj)

id: dob:

percentage:

sem

using

Required staters in project :: spring batch, mysql driver, lombok api, jdbc api

as

as

as

as

based

By RowMapper either seperate class or inner class or anonymous inner class or LAMDA expression inner class we can convert each record coming to RS object to given Model class object.

// Exam ResultRowMapper.java

package com.nt.mapper;

import java.sql.ResultSet;

(As Normal class)

import java.sql.SQLException;

import org.springframework.jdbc.core.RowMapper;

import com.nt.model.Exam Result;

public class Exam ResultRowMapper implements RowMapper<Exam Result> {

@Override

Callback method that executes automatically to give those many Model class objs as many db table recrods

public Exam Result mapRow(ResultSet rs, int rowNum) throws SQLException { return new Exam Result(rs.getInt(1),

}

}

for

rs.getDate(2),

rs.getDouble(3), rs.getInt(4));

syntax lamda based Interface implementation class obj creation

<interface name> ref= (<params>)-> { body of the method }

Lamda based Anonymous inner class as the impl class for RowMapper(1)

RowMapper<Exam Result> mapper= (rs, rowNum)->{ return new Exam Result(rs.getInt(1),

}

or

rs.getDate(2), rs.getDouble(3), rs.getInt(4));

RowMapper<Exam Result> mapper= (rs,rowNum)-> new Exam Result(rs.getInt(1),

rs.getDate(2),

rs.getDouble(3), rs.getInt(4));

RowMapper<T>

|---> public <T> mapRow(ResultSet rs,

int rowNum)thorw SQLExcepption (Functional interface becoz it is having only one method delcaration)

=> if the interface is functonal interface.. we can develop lamda expression based anonymous inner class as the implementation class for the funcational interface

=> interface with one abstract method directly or indirectly is called Functional interface => Interface with no methods and provides special runtime capabilties to the impl class object is called Marker Interface

In spring batch programming

```
reader.setRowMapper((rs,rowNumber)->new Exam Result(rs.getInt(1),
```

JdbcCursorItemReader

ob

holds the

ResultSet obj record number

having record

In this code, following operations are performed

```
rs.getDate(2),
rs.getDouble(3), rs.getInt(4)));
```

a) Anonymous inner class is created implementing RowMapper<T>

(b) In that inner class mapRow(rs, rowNumber) is

implemented having logic to copy ResultSet(rs) record to

Model class object (Exam Result obj)

(c) Anonymous inner class object is created and passed

to reader.setRowMapper(-)as argument value.

Best

JdbcCursorItemReader<T> sample code

=================================

```
@Bean
In BatchConfig.java
public JdbcCursorItemReader<Exam Result> createReader(){
//create object
JdbcCursorItemReader<Exam Result> reader=new JdbcCursorItemReader<>();
// specify DataSoruce
reader.setDataSource(ds);
// specify SQL Query
reader.setSql("SELECT ID, DOB, PERCENTAGE,SEMESTER FROM EXAM_RESULT"); //specify RowMapper
//reader.setRowMapper(new Exam ResultRowMapper());
reader.setRowMapper((rs,rowNumber)->new Exam Result(rs.getInt(1),
rs.getDate(2),
rs.getDouble(3), rs.getInt(4)));
}
return reader;
```

(or)

```
public JdbcCursorItemReader<Exam Result> createReader(){
//create and return object
return new JdbcCursorItemReaderBuilder<Exam Result>()
.dataSource(ds)
.sql("SELECT ID,DOB, PERCENTAGE,SEMESTER FROM EXAM_RESULT") .beanRowMapper(Exam
Result.class) // Internally use BeanPropertyRowMapper
.build();
}
//writer
@Bean
// to covert the record of RS to given Model class obj // but db table col names and model class properties
shoud match
public FlatFileItemWriter<Exam Result> createWriter(){
FlatFileItemWriter<Exam Result> writer=new FlatFileItem Writer<>(); //set logical name
// writer.setName("writer-csv");
//specify the destination csv file location
//writer.setResource(new ClassPath Resource("classpath:topbrains.csv"));
writer.setResource(new FileSystemResource("e:\\csvs\\topbrains.csv"));
// specify LineAggregator by supplying delimeter and Field Extractor
writer.setLineAggregator(new DelimitedLineAggregator<>() {{
//delimeter
setDelimiter(",");
//field extractor
setField Extractor(new BeanWrapperField Extractor<>() {{
//specify names to extracted field values.
setNames(new String[] {"id", "dob","percentage","semester"});
}});
}});
return writer;
}
(or)
@Bean
Example code
===============
<
>
public FlatFileItem Writer<Exam Result> createWriter(){ return new FlatFileItemWriterBuilder<Exam Result>()
```

.name("writer")

.resource(new FileSystemResource("TopBrains.csv")) .lineSeparator("\r\n")

.delimited().delimiter(",")

.names("id", "dob","percentage","semester") .build();

}

Model class

=============

BootBatchProj04-DBToCSV- SpringBoot3.x [boot]

>

src/main/java

#com.nt

> BootBatchProj04DbToCsvApplication.java

com.nt.config

> BatchConfig.java

com.nt.listener

> #com.nt.model

<

#com.nt.processor

import java.util.Date;

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

@Data

@NoArgsConstructor

<

com.nt.runners

JobLaunch TestRunner.java

src/main/resources

application.properties

src/test/java

JRE System Library [JavaSE-17]

Maven Dependencies

target/generated-sources/annotations target/generated-test-sources/test-annotations

>

>

>

src

>

target

HELP.md

mvnw

mvnw.cmd

pom.xml

TopBrains.csv

}

@AllArgsConstructor

public class Exam Result {

private Integer id;

private Date dob; private Float percentage;

private Integer semester;

}

```
/*@Bean(name="ffiw")
public FlatFileItem Writer<Exam Result> createWriter(){
//create writer object
FlatFileItemWriter<Exam Result> writer-new FlatFileItemWriter<Exam Result>();
//specify the resource
writer.setResource(new ClassPathResource("TopStudents.csv"));
(or)
//create FiledExatractor (To get values from Model class object)
BeanWrapperField Extractor<Exam Result> extractor=new BeanWrapperField Extractor<Exam Result>();
extractor.setNames(new String[] {"id", "dob","semester","percentage"});
//create Line Aggeregator (comblines everything into csv file lines)
DelimitedLineAggregator<Exam Result> aggregator=new DelimitedLineAggregator<Exam Result>();
aggregator.setDelimiter(",");
aggregator.setField Extractor(extractor);
//set LineAggregator to Writer obj
}*/
```

Item Processor class

========

======

```
package com.nt.processor;
import org.springframework.batch.item.ItemProcessor;
import org.springframework.stereotype.Component;
import com.nt.model.Exam Result;
@Component
writer.setLineAggregator(aggregator);
```

```
return writer;
public class Exam ResultItemProcessor implements ItemProcessor<Exam Result, Exam Result> {
@Override
public Exam Result process(Exam Result item) throws Exception {
if(item.getPercentage()>=90.0f) {
return item;
}
return null;
}
```

Listener class

==============

```
package com.nt.listener;
import java.util.Date;
import org.springframework.batch.core.Job Execution;
import org.springframework.batch.core.Job ExecutionListener;
import org.springframework.stereotype.Component;
@Component("jobListener")
public class JobMonitoringListener implements Job ExecutionListener {
private long start, end;
@Override
public void beforeJob(Job Execution jobExecution) {
System.out.println("JobMonitoringListener:: Job Started at::"+new Date());
start=System.currentTimeMillis();
}
@Override
public void afterJob(JobExecution jobExecution) {
}
}
System.out.println("Job Exit status ::"+job Execution.getExitStatus()+" at-->"+ new Date());
end=System.currentTimeMillis();
System.out.println("Job Excution time is ::"+(end-start)+" ms");
```

BatchConfig.java

===============

```
package com.nt.config;
import javax.sql.DataSource;
import org.springframework.batch.core.Job; import org.springframework.batch.core.Step; import
org.springframework.batch.core.job.builder.JobBuilder; import
org.springframework.batch.core.launch.support.RunIdIncrementer; import
org.springframework.batch.core.repository.JobRepository; import
org.springframework.batch.core.step.builder.StepBuilder; import
org.springframework.batch.item.database.JdbcCursorItemReader; import
org.springframework.batch.item.database.builder.JdbcCursorItem ReaderBuilder;
import org.springframework.batch.item.file.FlatFileItemWriter;
import org.springframework.batch.item.file.builder.FlatFileItem WriterBuilder;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean; import org.springframework.context.annotation.Configuration;
import org.springframework.core.io.FileSystem Resource;
import org.springframework.transaction. Platform Transaction Manager;
import com.nt.listener.JobMonitoringListener;
import com.nt.model.Exam Result;
import com.nt.processor.Exam ResultItemProcessor;
@Configuration
public class BatchConfig {
@Autowired
private JobMonitoringListener listener;
@Autowired
private DataSource ds;
@Autowired
private Exam ResultitemProcessor processor;
@Bean //reader
public JdbcCursorItemReader<Exam Result> createReader(){
return new JdbcCursorItemReaderBuilder<Exam Result>()
}
@Bean
.name("jdbc-reader")
.dataSource(ds)
.sql("SELECT ID,DOB, PERCENTAGE,SEMESTER FROM EXAM_RESULT")
.beanRowMapper(Exam Result.class)
.build();
public FlatFileItemWriter<Exam Result> createWriter(){
return new FlatFileItem WriterBuilder<Exam Result>()
.name("writer")
.resource(new FileSystemResource("TopBrains.csv"))
.lineSeparator("\r\n")
.delimited().delimiter(",")
.names("id", "dob","percentage","semester")
.build();
}
//Step obj
@Bean(name="step1")
public Step createStep1(Job Repository jobRepository, Platform Transaction Manager transaction Manager) {
return new StepBuilder("step1",jobRepository)
<Exam Result, Exam Result>chunk(3, transactionManager)
.reader(createReader())
.processor(processor)
.writer(createWriter())
.build();
}
//Job obj
@Bean(name="job1")
public Job createJob(Job Repository jobRepository, Step step1) {
return new JobBuilder("job1",jobRepository)
.incrementer(new RunIdIncrementer())
.listener(listener)
.start(step1)
.build();
}
```

Runner class

==========

```
package com.nt.runners;
import java.util.Date;
import org.springframework.batch.core.Job; import org.springframework.batch.core.Job Execution; 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 JobLaunch TestRunner implements CommandLineRunner { @Autowired
private JobLauncher launcher;
@Autowired
private Job job;
@Override
public void run(String... args) throws Exception {
try {
JobParameters params=new JobParametersBuilder().addDate("startDate", new Date()).toJobParameters();
JobExecution execution=launcher.run(job, params);
System.out.println("Job Execution Status ::"+execution.getExitStatus());
}
catch(Exception e) {
e.printStackTrace();
}
}
}
```

write to

`csv file`

TopBrains.csv