Batch processing 🡪 execution of series of the tasks or the jobs in group rather than processing a single task individually. We use it specially for handling large data or repetitive tasks. We can automate the batch processing by scripts or by software.

The tasks are grouped together into a batch and processed as unit.

We can use it in automated workflows, data processing or system maintenance

For scripting – bash or python or PowerShell

For jobs – Cron, task schedular

Batch, stream, micro batch processing 🡪

Batch processing -> it involves processing fixed amount of data all at once, minimizing user interaction. Data is collected over time, stored and processed in batch. Jobs are executed at scheduled intervals or triggered manually. Not a real time tasks, use batch processing

Stream processing -> it involves continuously processing data as it is generated or received at real time. data is processed, as it arrived before storing. Like Iot, fraud detection, or real time analytics

Micro batch processing -> it is hybrid of stream and batch processing. Processing is done in small batches at regular interval. Like processing log data of servers for detecting any anomalies

Spring boot batch processing 🡪

Jobs, jobs repo, steps, item reader, item writer, item processer.

JOBS REPO🡪 Job launcher –>> jobs –>> steps –>> item reader + item writer + item processer

Reader – read the data – src csv file

Processor – transforming or changing – logic

Writer – writes the data – destination a database (mysql , hypersql , posgresql)

Jobs and steps 🡪

A batch job includes steps or more than one step that encapsulated in unit of work

Chunk-oriented processing 🡪

Spring batch uses chunk-based processing where we can read process and write the data or we can say in manageable units called chunks.

Item reader – reads data from files , one at a time

Item processor – performs or add logic on every single item at a time

Item writer – writes processed data to destination

Job repositories – a kind of store that keeps track or manages histories for jobs – includes status , checkpoints

Tasklets – work or unit of execution in step , like for simple tasks sending mails

Listeners – kind of hook for monitoring jobs process , weather executed or not , handlings errors , before and after action of step

Partitioning – processing large datasets , for improve performance

ItemReader , create a class extending flatFileItemReader to read data from csv files

ItemProcessor, create a class implements itemReader class to execute logic

itemWriter , create a class which implements itemReader so write in databse in final

Implementation 🡪

Make a config folder-> classes we want:-

batchConfig

jobListner

in batch config 🡪

@configuration – indicating that object is a src of bean definations

Plus with this @bean – used to config specific instance

Job builder – it is used for helping in creating jobs with name, includes 2 parameters name and repo.

Name of the job , job repo to indications for job

After that there’s start() inside -> step – to execute step or sequence of step

Flow – to lead or execute a flow

And a decider

@EnableBatchProcessing - enables spring batch features

Now create a job 🡪 having repo name and step name

Inside it return job -> define steps to execute , like get , flow , start, build

Then create a step 🡪 use chunks here , means how u have to use data or read that particular objects

Reading from csv file

@bean

Public FlatFileItemReader <visitor> flatfileitemreader (@value(“${inputfile}”) Resource inputFile){

FlatFileItemReader <visitor>flatfileitemreader = new FlatFileItemReader<>();

flatfileitemreader.setname(“test123”);

flatfileitemreader.setLinesToSkip(1);

flatfileitemreader.setResource(inputFile);

flatfileitemreader.setLineMapper(linMap());

return flatfileitemreader;

}

@value(“${inputfile}”) 🡪 inject the value of property named inputfile from spring configuration (application.properties) into input file parameter.

Resource Inputfile 🡪 declare parameter named inputfile

FlatFileItemReader<Visitors> flatFileItemReader = new FlatFileItemReader<>();

This line creates a new instance of FlatFileItemReader.

flatFileItemReader.setLinesToSkip(1);

This line tells the reader to skip the first line of the input file. This is often used if the first line contains headers or metadata instead of actual data.

flatFileItemReader.setLineMapper(linMappe());

This line sets the LineMapper for the reader. The LineMapper is responsible for parsing each line of the input file into an object of type Visitors.

DefaultLineMapper<Visitors> defaultLineMapper = new DefaultLineMapper<>();

This line creates a new instance of DefaultLineMapper. This is the most common implementation of LineMapper in Spring Batch for flat files.

DelimitedLineTokenizer lineTokenizer = new DelimitedLineTokenizer();

This line creates a new instance of DelimitedLineTokenizer. This tokenizer splits each line of the input file based on a delimiter (usually a comma or tab).

lineTokenizer.setDelimiter(",");

This line sets the delimiter for the tokenizer to a comma (",").