**SpringBoot Batch Job Processing – 2024**

Spring batch is a framework processing large volumes of records, including transaction management, job processing statistics, and resource management.

**Pom.xml Relevant portion**

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-batch</artifactId>

</dependency>

Follow the steps mentioned below.

There are total **6 database tables** required to run the Spring Batch Processing

1. **Batch\_Job\_Execution**
2. **Batch\_Step\_Execution**
3. **Batch\_Job\_Execution\_Context**
4. **Batch\_Step\_Execution\_Context**
5. **Batch\_Job\_Execution\_Params**
6. **Batch\_Job\_Instance**

**Step-1: Run the following DDL script in the underlying database.**

**CREATE** **TABLE** BATCH\_JOB\_INSTANCE (

JOB\_INSTANCE\_ID BIGINT **NOT** **NULL** **PRIMARY** **KEY** ,

VERSION BIGINT ,

JOB\_NAME **VARCHAR**(100) **NOT** **NULL**,

JOB\_KEY **VARCHAR**(32) **NOT** **NULL**,

**constraint** JOB\_INST\_UN **unique** (JOB\_NAME, JOB\_KEY)

) ;

**CREATE** **TABLE** BATCH\_JOB\_EXECUTION (

JOB\_EXECUTION\_ID BIGINT **NOT** **NULL** **PRIMARY** **KEY** ,

VERSION BIGINT ,

JOB\_INSTANCE\_ID BIGINT **NOT** **NULL**,

CREATE\_TIME **TIMESTAMP** **NOT** **NULL**,

START\_TIME **TIMESTAMP** **DEFAULT** **NULL** ,

END\_TIME **TIMESTAMP** **DEFAULT** **NULL** ,

STATUS **VARCHAR**(10) ,

EXIT\_CODE **VARCHAR**(2500) ,

EXIT\_MESSAGE **VARCHAR**(2500) ,

LAST\_UPDATED **TIMESTAMP**,

**constraint** JOB\_INST\_EXEC\_FK **foreign** **key** (JOB\_INSTANCE\_ID)

**references** BATCH\_JOB\_INSTANCE(JOB\_INSTANCE\_ID)

) ;

**CREATE** **TABLE** BATCH\_JOB\_EXECUTION\_PARAMS (

JOB\_EXECUTION\_ID BIGINT **NOT** **NULL** ,

PARAMETER\_NAME **VARCHAR**(100) **NOT** **NULL** ,

PARAMETER\_TYPE **VARCHAR**(100) **NOT** **NULL** ,

PARAMETER\_VALUE **VARCHAR**(2500) ,

IDENTIFYING **CHAR**(1) **NOT** **NULL** ,

**constraint** JOB\_EXEC\_PARAMS\_FK **foreign** **key** (JOB\_EXECUTION\_ID)

**references** BATCH\_JOB\_EXECUTION(JOB\_EXECUTION\_ID)

) ;

**CREATE** **TABLE** BATCH\_STEP\_EXECUTION (

STEP\_EXECUTION\_ID BIGINT **NOT** **NULL** **PRIMARY** **KEY** ,

VERSION BIGINT **NOT** **NULL**,

STEP\_NAME **VARCHAR**(100) **NOT** **NULL**,

JOB\_EXECUTION\_ID BIGINT **NOT** **NULL**,

CREATE\_TIME **TIMESTAMP** **NOT** **NULL**,

START\_TIME **TIMESTAMP** **DEFAULT** **NULL** ,

END\_TIME **TIMESTAMP** **DEFAULT** **NULL** ,

STATUS **VARCHAR**(10) ,

COMMIT\_COUNT BIGINT ,

READ\_COUNT BIGINT ,

FILTER\_COUNT BIGINT ,

WRITE\_COUNT BIGINT ,

READ\_SKIP\_COUNT BIGINT ,

WRITE\_SKIP\_COUNT BIGINT ,

PROCESS\_SKIP\_COUNT BIGINT ,

ROLLBACK\_COUNT BIGINT ,

EXIT\_CODE **VARCHAR**(2500) ,

EXIT\_MESSAGE **VARCHAR**(2500) ,

LAST\_UPDATED **TIMESTAMP**,

**constraint** JOB\_EXEC\_STEP\_FK **foreign** **key** (JOB\_EXECUTION\_ID)

**references** BATCH\_JOB\_EXECUTION(JOB\_EXECUTION\_ID)

) ;

**CREATE** **TABLE** BATCH\_STEP\_EXECUTION\_CONTEXT (

STEP\_EXECUTION\_ID BIGINT **NOT** **NULL** **PRIMARY** **KEY**,

SHORT\_CONTEXT **VARCHAR**(2500) **NOT** **NULL**,

SERIALIZED\_CONTEXT TEXT ,

**constraint** STEP\_EXEC\_CTX\_FK **foreign** **key** (STEP\_EXECUTION\_ID)

**references** BATCH\_STEP\_EXECUTION(STEP\_EXECUTION\_ID)

) ;

**CREATE** **TABLE** BATCH\_JOB\_EXECUTION\_CONTEXT (

JOB\_EXECUTION\_ID BIGINT **NOT** **NULL** **PRIMARY** **KEY**,

SHORT\_CONTEXT **VARCHAR**(2500) **NOT** **NULL**,

SERIALIZED\_CONTEXT TEXT ,

**constraint** JOB\_EXEC\_CTX\_FK **foreign** **key** (JOB\_EXECUTION\_ID)

**references** BATCH\_JOB\_EXECUTION(JOB\_EXECUTION\_ID)

) ;

**CREATE** SEQUENCE BATCH\_STEP\_EXECUTION\_SEQ MAXVALUE 9223372036854775807 **NO** CYCLE;

**CREATE** SEQUENCE BATCH\_JOB\_EXECUTION\_SEQ MAXVALUE 9223372036854775807 **NO** CYCLE;

**CREATE** SEQUENCE BATCH\_JOB\_SEQ MAXVALUE 9223372036854775807 **NO** CYCLE;

**Step-2: Mention the following in application.properties file**

spring.datasource.url=jdbc:postgresql://localhost:5432/c3\_test\_db\_admin?currentSchema=public

spring.datasource.username=postgres

spring.datasource.password=Abcd@1234

spring.jpa.properties.hibernate.default\_schema=public

spring.jpa.database=POSTGRESQL

spring.datasource.driverClassName=org.postgresql.Driver

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

spring.jpa.properties.hibernate.id.db\_structure\_naming\_strategy=legacy

# Batch Job Related

spring.batch.job.enabled=false 🡸 Important

#Batch Chunk Size, chunk size value to be same as production

batch.chunk.size=1 🡸 Important

**Step-3: Create a main class like this.**

**@EnableBatchProcessing 🡸 Important**

@SpringBootApplication

**public** **class** SpringBatchJobApp {

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringBatchJobApp.**class**, args);

}

}

**Step-4: Create a Batch Job Configuration**

@Configuration

**public** **class** BatchJobConfig {

**@Bean(name = "appImportJob")**

**public** Job configureImportJob(JobRepository jobRepository,

@Qualifier("updateStep1") **Step updateStep1**, @Qualifier("insertStep2") **Step insertStep2**) {

**Job job = new JobBuilder("appImportJob", jobRepository)**

**.incrementer(new RunIdIncrementer()) 🡨 Optional**

**.start(updateStep1).next(insertStep2)**

**.build();**

// .listener(sfdcJobCompletionListener()) // Later

**return** job;

}

}

**Step-5: Create Batch Step Configuration**

@Configuration

**public** **class** BatchStepConfig {

@Value("${batch.chunk.size}")

**private** String batchChunkSize;

@Autowired

**private** UpdateItemReader1 updateItemReader1;

@Autowired

**private** ItemProcessor<? **super** Object, ? **extends** Object> updateItemProcessor;

@Autowired

**private** UpdateItemWriter updateItemWriter;

@Autowired

**private** InsertItemReader insertItemReader;

@Autowired

**private** InsertItemReadListener insertItemReadListener;

@Autowired

**private** InsertItemProcessor insertItemProcessor;

@Autowired

**private** InsertItemProcessListener insertItemProcessListener;

@Autowired

**private** InsertItemWriter insertItemWriter;

@Autowired

**private** InsertItemWriteListener insertItemWriteListener;

@Bean("updateStep1")

**public** Step updateStep1(JobRepository jobRepository, PlatformTransactionManager transactionManager) {

Step step1 = **new** StepBuilder("updateStep1", jobRepository)

.chunk(Integer.*parseInt*(batchChunkSize), transactionManager)

.faultTolerant()

.skip(Exception.**class**)

.skipLimit(9999999)

// .listener(sfdcOpportunitySkipListener)

.reader(updateItemReader1)

.processor(updateItemProcessor)

.writer(updateItemWriter)

.build();

**return** step1;

}

@Bean("insertStep2")

**public** Step insertStep2(JobRepository jobRepository, PlatformTransactionManager transactionManager) {

**return** **new** StepBuilder("insertStep2", jobRepository)

.chunk(Integer.*parseInt*(batchChunkSize), transactionManager)

.reader(insertItemReader)

.listener(insertItemReadListener)

.processor(insertItemProcessor)

.listener(insertItemProcessListener)

.writer(insertItemWriter)

.listener(insertItemWriteListener)

.build();

}

}

**Step-6: Create Corresponding Reader, Processor and Writer classes for Batch Step Configuration**

Few examples are given below.

**Item Reader Implementation**

@Component("updateItemReader1")

**public** **class** UpdateItemReader1 **implements** ItemReader<Person> {

List<Person> personList = **new** ArrayList<>();

**public** UpdateItemReader1() {

**for** (**int** i = 0; i < 30; i++) {

personList.add(**new** Person(i, "Name-" + i, "Data-" + i));

}

}

@Override

**public** Person read() **throws** Exception, UnexpectedInputException, ParseException, NonTransientResourceException {

**if**(!personList.isEmpty()) {

**return** personList.remove(0);

}

**return** **null**;

}

}

**Item Processor Implementation**

@Component("updateItemProcessor")

**public** **class** UpdatetemProcessor **implements** ItemProcessor<Object, Object> {

@Override

**public** Person process(Object obj) **throws** Exception {

System.***out***.println("--------- Inside Update Item Processor in Update Step -------");

Person item = (Person) obj;

System.***out***.println("Processing Item: "+item);

System.***out***.println("--------- Inside Update Item Processor in Update Step -------"); **return** item;

}

}

**Item Writer Implementation**

@Component("updateItemWriter")

**public** **class** UpdateItemWriter **implements** ItemWriter<Object> {

@Override

**public** **void** write(Chunk<? **extends** Object> chunk) **throws** Exception {

System.***out***.println("------------ WRITING ITEMS in Update Step -------------------");

System.***out***.println("Chunk Items: "+chunk.getItems());

chunk.getItems().stream().forEach( value -> System.***out***.println("Value to write--->"+value));

System.***out***.println("------------ WRITING ITEMS in Update Step -------------------");

}

}

**Item Listener Implementation**

@Component("insertItemProcessListener")

**public** **class** InsertItemProcessListener {

**@BeforeProcess**

**public** **void** beforeProcess(Person item) {

System.***out***.println("InsertItemProcessListener Before Processing person item: "+item);

}

**@AfterProcess**

**public** **void** afterProcess(Person item, Object result) {

System.***out***.println("InsertItemProcessListener After Processing person item: "+item);

}

**@OnProcessError**

**public** **void** onProcessError(Person item, Exception e) {

System.***out***.println("Listener onProcessError: Processing person item: "+item);

}

}

For a Listener, you have to implement the following manner

**@BeforeProcess**

**@AfterProcess**

**@OnProcessError**

**AutoRun Class is given below**

@Component

**public** **class** AutoRun {

@Autowired

@Qualifier("appImportJob")

**private** Job importBatchJob;

@Autowired

**private** **JobLauncher jobLauncher**;

@EventListener(ApplicationReadyEvent.**class**)

**public** **void** run() {

System.***out***.println("Started running the application ...");

**launchJob();**

}

**private** JobExecution launchJob() {

**JobParameters jobParameters = new JobParametersBuilder()**

**.addDate("date", new Date()).toJobParameters();**

JobExecution jobExecution = **null**;

**try** {

**jobExecution = jobLauncher.run(importBatchJob, jobParameters);**

} **catch** (IllegalArgumentException | JobParametersInvalidException | UnexpectedJobExecutionException

| JobExecutionAlreadyRunningException | JobRestartException | JobInstanceAlreadyCompleteException e) {

System.***err***.println("Exception in running the batch job execution ..."+e.getMessage());

e.printStackTrace();

}

**if** (jobExecution == **null**) {

System.***err***.println("Failed to execute job");

**return** jobExecution;

}

**return** jobExecution;

}

}

**Useful Queries**

-- To see the job execution details

**select** bji.job\_instance\_id, bji.job\_name, bje.start\_time, bje.end\_time, bje.status, bje.exit\_code, bje.exit\_message, bje.last\_updated

**from** BATCH\_JOB\_INSTANCE bji, BATCH\_JOB\_EXECUTION bje

**where** bji.job\_instance\_id = bje.job\_instance\_id;

-- To see the step execution details

**select** bje.job\_execution\_id, bse.step\_name, bse.start\_time, bse.end\_time, bse.status, bse.exit\_code

**from** BATCH\_JOB\_EXECUTION bje, BATCH\_STEP\_EXECUTION bse

**where** bje.job\_execution\_id = bse.job\_execution\_id;

**How to Prepare Configuration Precisely**

**Create a Batch Job Configuration**

**@Configuration**

**public class BatchJobConfig {**

**@Bean(name = "appJob")**

**public Job configureJob(JobRepository jobRepo,**

**@Qualifier("step1") Step step1,**

**@Qualifier("istep2") Step step2) {**

**Job job = new JobBuilder("appJob", jobRepo)**

**.incrementer(new RunIdIncrementer())**

**.start(step1)**

**.next(step2)**

**.build();**

**return job;**

**}**

**}**

**Create Batch Step Configuration**

**@Configuration**

**public class BatchStepConfig {**

**@Value("${batch.chunk.size}")**

**private String batchChunkSize;**

**@Bean("step1")**

**public Step step1(JobRepository jobRepo, PlatformTransactionManager txnMgr) {**

**Step step1 = new StepBuilder("step1", jobRepo)**

**.chunk(Integer.parseInt(batchChunkSize), txnMgr)**

**.reader(itemReader1)**

**.processor(itemProcessor)**

**.writer(itemWriter)**

**.build();**

**return step1;**

**}**

**@Bean("step2")**

**public Step step2(JobRepository jobRepo, PlatformTransactionManager txnMgr) {**

**return new StepBuilder("insertStep2", jobRepo)**

**.chunk(Integer.parseInt(batchChunkSize), txnMgr)**

**.reader(insertItemReader)**

**.listener(insertItemReadListener)**

**.processor(insertItemProcessor)**

**.listener(insertItemProcessListener)**

**.writer(insertItemWriter)**

**.listener(insertItemWriteListener)**

**.build();**

**}**

**}**

**How to run the job**

**@Autowired**

**@Qualifier("appJob ")**

**private Job appJob;**

**@Autowired**

**private JobLauncher jobLauncher;**

**private JobExecution launchJob() {**

**JobParameters jobParameters = new JobParametersBuilder()**

**.addDate("date", new Date()).toJobParameters();**

**JobExecution jobExecution = null;**

**jobExecution = jobLauncher.run(appJob, jobParameters);**

**return jobExecution;**

**}**

**What is .incrementer(new RunIdIncrementer()) in Spring Batch Processing ?**

In Spring Batch processing, .**incrementer(new RunIdIncrementer())** is used to add an incrementer to a job. The RunIdIncrementer is a built-in implementation that generates a unique run ID for each job run by incrementing a counter.

 **Purpose**: The incrementer ensures that each execution of a job is unique, even if the job parameters are the same.

 **Implementation**: RunIdIncrementer increments a counter that is appended to the job parameters.

 **Usage**: It helps in identifying and distinguishing different runs of the same job, which is crucial for job execution tracking and handling.