Spring Batch CSV to MySQL Project Documentation

# 1. Introduction

This project demonstrates the use of Spring Batch to read large CSV files asynchronously and write the data into a MySQL database ('customersinfo'). The Spring Boot version used is 2.6.7.

# 2. Requirements

- Spring Boot 2.6.7  
- Spring Batch  
- MySQL  
- CSV File (customers.csv)  
- Java 11+

# 3. Setup Instructions

## 3.1 application.properties

spring.application.name=springbatch  
server.port=8989  
spring.datasource.url=jdbc:mysql://localhost:3306/customerinfo  
spring.datasource.username=root  
spring.datasource.password=root  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect  
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver  
spring.batch.initialize-schema=ALWAYS  
#disabled job run at startup  
spring.batch.job.enabled=false

## 3.2 JobController.java

@RestController  
@RequestMapping("/jobs")  
public class JobController {  
  
 @Autowired  
 private JobLauncher jobLauncher;  
  
 @Autowired  
 private Job job;  
  
 @PostMapping("/importCustomers")  
 public void importCSVToDbJob() {  
 JobParameters jobParameters = new JobParametersBuilder()  
 .addLong("startAt", System.currentTimeMillis()).toJobParameters();  
  
 try {  
 jobLauncher.run(job, jobParameters);  
 } catch (Exception e) {  
 throw new RuntimeException(e);  
 }  
 }  
}

## 3.3 SpringBatchConfig.java

@Configuration  
@EnableBatchProcessing  
@AllArgsConstructor  
public class SpringBatchConfig {  
  
 private JobBuilderFactory jobBuilderFactory;  
 private StepBuilderFactory stepBuilderFactory;  
 private CustomerRepository customerRepository;  
  
 @Bean  
 public FlatFileItemReader<Customer> reader() {  
 FlatFileItemReader<Customer> itemReader = new FlatFileItemReader<>();  
 itemReader.setResource(new FileSystemResource("src/main/resources/customers.csv"));  
 itemReader.setName("csvReader");  
 itemReader.setLinesToSkip(1);  
 itemReader.setLineMapper(lineMapper());  
 return itemReader;  
 }  
  
 private LineMapper<Customer> lineMapper() {  
 DefaultLineMapper<Customer> lineMapper = new DefaultLineMapper<>();  
 DelimitedLineTokenizer lineTokenizer = new DelimitedLineTokenizer();  
 lineTokenizer.setDelimiter(",");  
 lineTokenizer.setStrict(false);  
 lineTokenizer.setNames("id", "firstName", "lastName", "email", "gender", "contactNo", "country", "dob");  
  
 BeanWrapperFieldSetMapper<Customer> mapper = new BeanWrapperFieldSetMapper<>();  
 mapper.setTargetType(Customer.class);  
 lineMapper.setLineTokenizer(lineTokenizer);  
 lineMapper.setFieldSetMapper(mapper);  
  
 return lineMapper;  
 }  
  
 @Bean  
 public CustomerProcessor processor() {  
 return new CustomerProcessor();  
 }  
  
 @Bean  
 public RepositoryItemWriter<Customer> writer() {  
 RepositoryItemWriter<Customer> writer = new RepositoryItemWriter<>();  
 writer.setRepository(customerRepository);  
 writer.setMethodName("save");  
 return writer;  
 }  
  
 @Bean  
 public Step step1() {  
 return stepBuilderFactory.get("csv-step").<Customer, Customer>chunk(10)  
 .reader(reader())  
 .processor(processor())  
 .writer(writer())  
 .taskExecutor(taskExecutor())  
 .build();  
 }  
  
 @Bean  
 public Job runJob() {  
 return jobBuilderFactory.get("importCustomers")  
 .flow(step1())  
 .end()  
 .build();  
 }  
  
 @Bean  
 public TaskExecutor taskExecutor() {  
 SimpleAsyncTaskExecutor asyncTaskExecutor = new SimpleAsyncTaskExecutor();  
 asyncTaskExecutor.setConcurrencyLimit(10);  
 return asyncTaskExecutor;  
 }  
}

## 3.4 CustomerProcessor.java

public class CustomerProcessor implements ItemProcessor<Customer, Customer> {  
 @Override  
 public Customer process(Customer customer) throws Exception {  
 if (customer.getCountry().equals("United States")) {  
 return customer;  
 } else {  
 return null;  
 }  
 }  
}

# 4. Execution Flow

1. The controller exposes an endpoint '/jobs/importCustomers' that triggers the batch job.  
2. The SpringBatchConfig defines the step to read the CSV file asynchronously, process records, and write them to MySQL.  
3. The CSV records are read using FlatFileItemReader and mapped to Customer entities.  
4. The CustomerProcessor filters out non-United States customers before writing to the database.  
5. The RepositoryItemWriter writes valid customers to the MySQL database.

# 5. Conclusion

This project showcases an example of asynchronous batch processing using Spring Batch. It efficiently reads large CSV files and processes them concurrently, filtering and saving valid customer data to the MySQL database.