// src/main/java/com/example/batch/ZipBatchApplication.java

package com.example.batch;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class ZipBatchApplication {

public static void main(String[] args) {

SpringApplication.run(ZipBatchApplication.class, args);

}

}

// src/main/java/com/example/batch/entity/Order.java

package com.example.batch.entity;

import jakarta.persistence.\*;

@Entity

@Table(name = "orders")

public class Order {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long orderId;

private String status;

// Getters and setters

public Long getOrderId() {

return orderId;

}

public void setOrderId(Long orderId) {

this.orderId = orderId;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

}

// src/main/java/com/example/batch/model/ZipResult.java

package com.example.batch.model;

public class ZipResult {

private Long orderId;

private String zipFileAPath;

private String zipFileBPath;

public ZipResult(Long orderId, String zipFileAPath, String zipFileBPath) {

this.orderId = orderId;

this.zipFileAPath = zipFileAPath;

this.zipFileBPath = zipFileBPath;

}

public Long getOrderId() {

return orderId;

}

public String getZipFileAPath() {

return zipFileAPath;

}

public String getZipFileBPath() {

return zipFileBPath;

}

}

// src/main/java/com/example/batch/reader/OrderItemReader.java

package com.example.batch.reader;

import com.example.batch.entity.Order;

import jakarta.persistence.EntityManagerFactory;

import org.springframework.batch.item.database.JpaPagingItemReader;

import org.springframework.batch.item.database.builder.JpaPagingItemReaderBuilder;

import org.springframework.context.annotation.Bean;

import org.springframework.stereotype.Component;

@Component

public class OrderItemReader {

@Bean

public JpaPagingItemReader<Order> reader(EntityManagerFactory entityManagerFactory) {

return new JpaPagingItemReaderBuilder<Order>()

.name("orderItemReader")

.entityManagerFactory(entityManagerFactory)

.queryString("SELECT o FROM Order o WHERE o.status = 'RECEIVED'")

.pageSize(10)

.build();

}

}

// src/main/java/com/example/batch/config/BatchConfig.java

package com.example.batch.config;

import com.example.batch.entity.Order;

import com.example.batch.model.ZipResult;

import com.example.batch.processor.OrderProcessor;

import com.example.batch.reader.OrderItemReader;

import com.example.batch.tasklet.MoveZipTasklet;

import com.example.batch.tasklet.UpdateOrderStatusTasklet;

import com.example.batch.writer.ZipWriter;

import jakarta.persistence.EntityManagerFactory;

import org.springframework.batch.core.Job;

import org.springframework.batch.core.Step;

import org.springframework.batch.core.configuration.annotation.EnableBatchProcessing;

import org.springframework.batch.core.configuration.annotation.JobBuilderFactory;

import org.springframework.batch.core.configuration.annotation.StepBuilderFactory;

import org.springframework.batch.item.database.JpaPagingItemReader;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

@Configuration

@EnableBatchProcessing

public class BatchConfig {

@Bean

public OrderProcessor orderProcessor() {

return new OrderProcessor();

}

@Bean

public ZipWriter zipWriter() {

return new ZipWriter();

}

@Bean

public Step chunkStep(StepBuilderFactory stepBuilderFactory,

JpaPagingItemReader<Order> reader,

OrderProcessor processor,

ZipWriter writer) {

return stepBuilderFactory.get("chunkStep")

.<Order, ZipResult>chunk(10)

.reader(reader)

.processor(processor)

.writer(writer)

.build();

}

@Bean

public Step moveZipStep(StepBuilderFactory stepBuilderFactory, MoveZipTasklet moveZipTasklet) {

return stepBuilderFactory.get("moveZipStep")

.tasklet(moveZipTasklet)

.build();

}

@Bean

public Step updateOrderStatusStep(StepBuilderFactory stepBuilderFactory,

UpdateOrderStatusTasklet updateOrderStatusTasklet) {

return stepBuilderFactory.get("updateOrderStatusStep")

.tasklet(updateOrderStatusTasklet)

.build();

}

@Bean

public Job zipHandlingJob(JobBuilderFactory jobBuilderFactory,

Step chunkStep,

Step moveZipStep,

Step updateOrderStatusStep) {

return jobBuilderFactory.get("zipHandlingJob")

.start(chunkStep)

.next(moveZipStep)

.next(updateOrderStatusStep)

.build();

}

}

// src/main/java/com/example/batch/processor/OrderProcessor.java

package com.example.batch.processor;

import com.example.batch.entity.Order;

import com.example.batch.model.ZipResult;

import org.springframework.batch.item.ItemProcessor;

import java.nio.file.Paths;

public class OrderProcessor implements ItemProcessor<Order, ZipResult> {

@Override

public ZipResult process(Order order) {

String zipFileA = "/source/zips/A\_" + order.getOrderId() + ".zip";

String zipFileB = "/source/zips/B\_" + order.getOrderId() + ".zip";

return new ZipResult(order.getOrderId(), zipFileA, zipFileB);

}

}

// src/main/java/com/example/batch/writer/ZipWriter.java

package com.example.batch.writer;

import com.example.batch.model.ZipResult;

import org.springframework.batch.item.ItemWriter;

import java.io.File;

import java.io.FileOutputStream;

import java.io.IOException;

import java.util.List;

import java.util.zip.ZipEntry;

import java.util.zip.ZipOutputStream;

public class ZipWriter implements ItemWriter<ZipResult> {

@Override

public void write(List<? extends ZipResult> items) throws IOException {

for (ZipResult result : items) {

File zipFile = new File(result.getZipFileAPath());

try (ZipOutputStream zipOut = new ZipOutputStream(new FileOutputStream(zipFile, true))) {

File pdfFile = new File("/source/pdfs/" + result.getOrderId() + ".pdf");

if (pdfFile.exists()) {

zipOut.putNextEntry(new ZipEntry(pdfFile.getName()));

zipOut.write(java.nio.file.Files.readAllBytes(pdfFile.toPath()));

zipOut.closeEntry();

}

}

}

}

}

// src/main/java/com/example/batch/tasklet/MoveZipTasklet.java

package com.example.batch.tasklet;

import org.springframework.batch.core.StepContribution;

import org.springframework.batch.core.scope.context.ChunkContext;

import org.springframework.batch.core.step.tasklet.Tasklet;

import org.springframework.batch.repeat.RepeatStatus;

import org.springframework.stereotype.Component;

import java.io.File;

import java.nio.file.Files;

import java.nio.file.Path;

import java.nio.file.StandardCopyOption;

@Component

public class MoveZipTasklet implements Tasklet {

@Override

public RepeatStatus execute(StepContribution contribution, ChunkContext chunkContext) throws Exception {

File sourceDir = new File("/source/zips");

File targetDir = new File("/target/zips");

if (!targetDir.exists()) {

targetDir.mkdirs();

}

for (File file : sourceDir.listFiles()) {

Files.move(file.toPath(), Path.of(targetDir.getAbsolutePath(), file.getName()), StandardCopyOption.REPLACE\_EXISTING);

}

return RepeatStatus.FINISHED;

}

}

// src/main/java/com/example/batch/tasklet/UpdateOrderStatusTasklet.java

package com.example.batch.tasklet;

import com.example.batch.repository.OrderRepository;

import org.springframework.batch.core.StepContribution;

import org.springframework.batch.core.scope.context.ChunkContext;

import org.springframework.batch.core.step.tasklet.Tasklet;

import org.springframework.batch.repeat.RepeatStatus;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Component;

@Component

public class UpdateOrderStatusTasklet implements Tasklet {

@Autowired

private OrderRepository orderRepository;

@Override

public RepeatStatus execute(StepContribution contribution, ChunkContext chunkContext) throws Exception {

orderRepository.updateOrderStatus("PROCESSED");

return RepeatStatus.FINISHED;

}

}

// src/main/java/com/example/batch/repository/OrderRepository.java

package com.example.batch.repository;

import com.example.batch.entity.Order;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.data.jpa.repository.Modifying;

import org.springframework.data.jpa.repository.Query;

import org.springframework.stereotype.Repository;

import org.springframework.transaction.annotation.Transactional;

@Repository

public interface OrderRepository extends JpaRepository<Order, Long> {

@Transactional

@Modifying

@Query("UPDATE Order o SET o.status = :status WHERE o.status = 'RECEIVED'")

void updateOrderStatus(String status);

}

// pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>zip-batch</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<properties>

<java.version>17</java.version>

<spring.boot.version>3.1.3</spring.boot.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>com.ibm.db2</groupId>

<artifactId>jcc</artifactId>

<version>11.5.7.0</version>

</dependency>

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>