File processing + maven project + hibernate(oracle DB) + kafka Producer

Pom.xml

<dependency>

<groupId>org.apache</groupId>

<artifactId>log4j</artifactId>

<version>1.2.16</version>

<scope>provided</scope>

</dependency>

<dependency>

<groupId>org.hibernate</groupId>

<artifactId>hibernate-core</artifactId>

<version>4.3.0.Final</version>

</dependency>

<dependency>

<groupId>com.oracle</groupId>

<artifactId>ojdbc</artifactId>

<version>14</version>

<scope>provided</scope>

</dependency>

**File Processing:**

**import** java.nio.file.Files;

**import** java.nio.file.Path;

**import** java.nio.file.Paths;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.stream.Collectors;

**import** java.util.stream.Stream;

**public** **class** FileProcessing {

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

// **TODO** Auto-generated method stub

String filename = "c://sachin/Lines.txt";

List<String> list = **new** ArrayList();

List<String[]> sublist = **new** ArrayList<String[]>();

**try**(Stream<String> stream = Files.*lines*(Paths.*get*(filename))){

list = stream

.filter(line -> !line.startsWith("Header"))

.map(String::toUpperCase)

.collect(Collectors.*toList*());

}**catch**(Exception e) {

e.printStackTrace();

}

sublist =list.stream()

.map(str->str.split("\\|"))

.collect(Collectors.*toList*());

**for**(String[] str: sublist)

System.***out***.println(str[4]);

System.***out***.println("inside the file");

list.forEach(System.***out***::println);

}

}

Hibernate Connection:

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<!DOCTYPE hibernate-configuration SYSTEM

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name=*"hibernate.dialect"*>org.hibernate.dialect.Oracle10gDialect</property>

<property name=*"hibernate.connection.driver\_class"*>oracle.jdbc.driver.OracleDriver </property>

<property name=*"hibernate.current\_session\_context\_class"*>thread</property>

<!-- List of XML mapping files -->

<mapping resource=*"GcpSareaTmpData.hbm.xml"*/>

</session-factory>

</hibernate-configuration>

***GcpSareaTmpData.hbm.xml***

<?xml version=*"1.0"* encoding=*"utf-8"*?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD//EN"

"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">

<hibernate-mapping>

<class name=*"com.att.pmoss.cti.GcpSareaTmpData"* table=*"GCPSAREATMP"*>

<meta attribute=*"class-description"*>

This class contains the dns, portname ,configuredPortSpeed and authorizedAllocationRatio

</meta>

<id name=*"ptnii"* column=*"PTNII"* type=*"string"*/>

<property name=*"portIdentifier"* column=*"PORTIDENTIFIER"* type=*"string"*/>

<property name=*"configuredPortSpeed"* column=*"CONFIGUREDPORTSPEED"* type=*"string"*/>

<property name=*"authorizedAllocationRatio"* column=*"AUTHORIZEDALLOCATIONRATIO"* type=*"string"*/>

<property name=*"reservationInEffectIndicator"* column=*"RESERVATIONINEFFECTINDICATOR"* type=*"string"*/>

<property name=*"weightFactor"* column=*"WEIGHTFACTOR"* type=*"string"*/>

<property name=*"portLock"* column=*"PORTLOCK"* type=*"string"*/>

<property name=*"protectionMode"* column=*"PROTECTIONMODE"* type=*"string"*/>

<property name=*"authorizedDynamicBw"* column=*"AUTHORIZEDDYNAMICBW"* type=*"string"*/>

<property name=*"espEnforcedMaxEvcCount"* column=*"ESPENFORCEDMAXEVCCOUNT"* type=*"string"*/>

</class>

<class name=*"com.att.pmoss.cti.GcpSareaCalculatedData"* table=*"OVERBOOKINGINTERFACES"*>

<meta attribute=*"class-description"*>

This class contains the dns, portname ,configuredPortSpeed, authorizedAllocationRatio,adjustedSpeed and speed

</meta>

<id name=*"ptnii"* column=*"DNS"* type=*"string"*/>

<property name=*"portIdentifier"* column=*"PORTIDENTIFIER"* type=*"string"*/>

<property name=*"configuredPortSpeed"* column=*"CONFIGUREDPORTSPEED"* type=*"string"*/>

<property name=*"authorizedAllocationRatio"* column=*"AUTHORIZEDALLOCATIONRATIO"* type=*"string"*/>

<property name=*"adjustedSpeed"* column=*"ADJUSTEDSPEED"* type=*"string"*/>

<property name=*"speed"* column=*"SPEED"* type=*"string"*/>

<property name=*"interfaceName"* column=*"INTERFACENAME"* type=*"string"*/>

<property name=*"reservationInEffectIndicator"* column=*"RESERVATIONINEFFECTINDICATOR"* type=*"string"*/>

<property name=*"weightFactor"* column=*"WEIGHTFACTOR"* type=*"string"*/>

<property name=*"portLock"* column=*"PORTLOCK"* type=*"string"*/>

<property name=*"protectionMode"* column=*"PROTECTIONMODE"* type=*"string"*/>

<property name=*"authorizedDynamicBw"* column=*"AUTHORIZEDDYNAMICBW"* type=*"string"*/>

<property name=*"espEnforcedMaxEvcCount"* column=*"ESPENFORCEDMAXEVCCOUNT"* type=*"string"*/>

</class>

</hibernate-mapping>

**import** org.hibernate.SessionFactory;

**import** org.hibernate.boot.registry.StandardServiceRegistry;

**import** org.hibernate.boot.registry.StandardServiceRegistryBuilder;

**import** org.hibernate.cfg.Configuration;

/\* #Topology support overbooking and sub-rated INLs

\* @Description : SAREA interfaces received from GCP/EDF and publish them in a new feed for consumption by vETL/Vertica

\* @Author : Sachin Chippalakatti<sc849m@intl.att.com>

\*

\* Following are the steps involved..

\* 1. Reading the tmp table data into List.

\* 2. Put all the DNS in to Map.

\* 3. Get the value of the speed from Device and Interface Table, put into Map<dns,Map<ifindex/desc,LIST[VENDOR, IFNAME, IFSPEED]>>

\* 4. Calculating the AdjustedSpeed - this is calculated as ConfiguredPortSpeed x 1000000 [units of b/s]

\* 5. Set the value of the speed to resulting list

\* 6. Write the resulting list to the DB table OVERBOOKINGINTERFACES.

\*/

**public** **class** OverbookingsSubRatingCalculation {

**private** **static** SessionFactory factory;

**private** **static** Logger logger = Logger.getLogger(OverbookingsSubRatingCalculation.**class**);

**private** OverbookingsSubRatingCalculation(){}

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

{

logger.info("Entering into sareaOverbookings:: OverbookingsSubRatingCalculationd");

Configuration configuration=**null**;

StandardServiceRegistry serviceRegistry=**null**;

**try**{

configuration = **new** Configuration().configure();

configuration.setProperty("hibernate.connection.url", System.getenv("JDBC\_URL"));

configuration.setProperty("hibernate.connection.username", System.getenv("USERNAME"));

configuration.setProperty("hibernate.connection.password", System.getenv("PASSWORD"));

serviceRegistry =

**new** StandardServiceRegistryBuilder().applySettings(configuration.getProperties()).build();

factory = configuration.buildSessionFactory(serviceRegistry);

logger.error("Created sessionFactory object......");

}**catch** (RuntimeException ex) {

logger.error("Failed to create sessionFactory object." + ex);

**return**;

}

//Save data in temp table

**public** **static** **void** addGcpSareaCalculatedDataToTable(SessionFactory factory, GcpSareaCalculatedData caldata){

Session session = factory.getCurrentSession();

Transaction tx = **null**;

**try**{

tx = session.beginTransaction();

session.save(caldata);

tx.commit();

}**catch** (HibernateException e) {

logger.info("Hibernate Exception while begining the transaction in method addGcpSareaCalculatedDataToTable" + e);

**if** (tx!=**null**)

tx.rollback();

e.printStackTrace();

}

}

//Read the data from temp table and process it

List <GcpSareaTmpData>gcpTmpData = **new** ArrayList<>();

List<Object[]> gcpTmpDatalist = OverbookingsSubRatingUtil.readingDataFromTmpTable(factory);

//Reading Data from table

**public** **static** List<Object[]> readingDataFromTmpTable(SessionFactory factory){

Session session = factory.getCurrentSession();

Transaction tx = **null**;

List<Object[]> gcpTmpDatalist = **null**;

**try**{

tx = session.beginTransaction();

org.hibernate.Query query = session.createSQLQuery(" Select \* FROM GCPSAREATMP");

gcpTmpDatalist = query.list();

tx.commit();

}**catch** (HibernateException e) {

logger.info("Hibernate Exception while begining the transaction in method readingDataFromTmpTable" + e);

**if** (tx!=**null**)

tx.rollback();

e.printStackTrace();

}

**return** gcpTmpDatalist;

}

//Rejected record writing to txt file.

**public** **static** **void** rejectedRecord(List<GcpSareaCalculatedData> gcpSareaCalculatedData)

{

Date date = **new** Date() ;

SimpleDateFormat dateFormat = **new** SimpleDateFormat("yyyyMMdd");

String path = "/feeds/incoming/gcp/";

String outputFile ="EDFSAREAFFD\_REJECTED\_RECORDS\_"+ dateFormat.format(date) + ".txt";

PrintWriter writer;

outputFile=System.getenv("CTI")+"//"+ path +outputFile;

**try** {

writer = **new** PrintWriter(outputFile, "UTF-8");

writer.println("DNS,PORTIDENTIFIER,CONFIGUREDPORTSPEED,AUTHORIZEDALLOCATIONRATIO");

**for**(GcpSareaCalculatedData caldata : gcpSareaCalculatedData){

writer.println(caldata.getPtnii() + "," + caldata.getPortIdentifier() + "," + caldata.getConfiguredPortSpeed() + "," + caldata.getAuthorizedAllocationRatio());

}

writer.close();

} **catch** (FileNotFoundException e) {

logger.error("File Not Found Exception for creating the rejecting the file" + e);

} **catch** (UnsupportedEncodingException e) {

logger.error("Failed in encoding the file" + e);

}

logger.info("Exiting rejectedRecord");

}

// Producer configuration

    @Bean

    public Map<String, Object> producerConfigs() {

        Map<String, Object> props =

                new HashMap<>(kafkaProperties.buildProducerProperties());

        props.put(ProducerConfig.KEY\_SERIALIZER\_CLASS\_CONFIG,

                StringSerializer.class);

        props.put(ProducerConfig.VALUE\_SERIALIZER\_CLASS\_CONFIG,

                JsonSerializer.class);

        return props;

    }

    @Bean

    public ProducerFactory<String, Object> producerFactory() {

        return new DefaultKafkaProducerFactory<>(producerConfigs());

    }

    @Bean

    public KafkaTemplate<String, Object> kafkaTemplate() {

        return new KafkaTemplate<>(producerFactory());

    }

    @Bean

    public NewTopic adviceTopic() {

        return new NewTopic(topicName, 3, (short) 1);

    }

}

Controller:

@RestController

public class HelloKafkaController {

    private static final Logger logger =

            LoggerFactory.getLogger(HelloKafkaController.class);

    private final KafkaTemplate<String, Object> template;

    private final String topicName;

    private final int messagesPerRequest;

    private CountDownLatch latch;

    public HelloKafkaController(

            final KafkaTemplate<String, Object> template,

            @Value("${tpd.topic-name}") final String topicName,

            @Value("${tpd.messages-per-request}") final int messagesPerRequest) {

        this.template = template;

        this.topicName = topicName;

        this.messagesPerRequest = messagesPerRequest;

    }

    @GetMapping("/hello")

    public String hello() throws Exception {

        latch = new CountDownLatch(messagesPerRequest);

        IntStream.range(0, messagesPerRequest)

                .forEach(i -> this.template.send(topicName, String.valueOf(i),

                        new PracticalAdvice("A Practical Advice", i))

                );

        latch.await(60, TimeUnit.SECONDS);

        logger.info("All messages received");

        return "Hello Kafka!";

    }

}