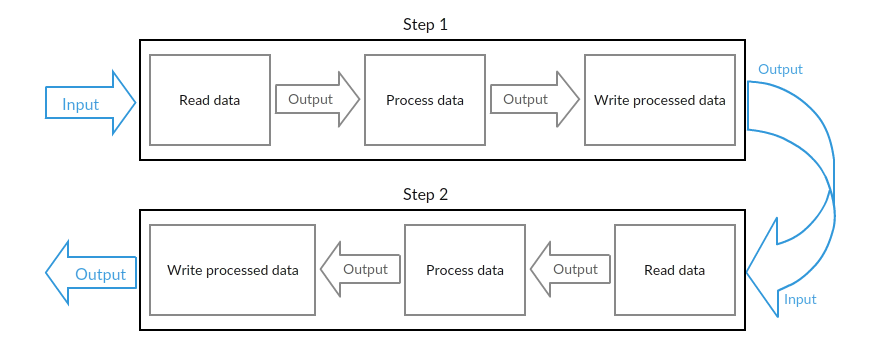
* **Spring Batch**

Spring Batch is a lightweight, comprehensive framework designed to facilitate development of robust batch applications. It also provides more advanced technical services and features that support extremely high volume and high-performance batch jobs through its optimization and partitioning techniques. Spring Batch builds upon the POJO-based development approach of the Spring Framework.

A batch job consists of one or more steps. Each step is responsible of completing one logical task. Every step reads input data, processes the input data, and writes the processed data to the configured output. If the batch job has more one step, the output of a step is often used as an input of the next step. 

**Batch processing** is a processing mode which involves execution of series of automated complex jobs without user interaction. A batch process handles bulk data and runs for a long time.

Several Enterprise applications require to process huge data to perform operations involving −

* Time-based events such as periodic calculations.
* Periodic applications that are processed repetitively over large datasets.
* Applications that deals with processing and validation of the data available in a transactional manner.

Therefore, batch processing is used in enterprise applications to perform such transactions.

**What is Spring Batch**

Spring batch is a **lightweight framework** which is used to develop **Batch Applications** that are used in Enterprise Applications.

In addition to bulk processing, this framework provides functions for −

* Including logging and tracing
* Transaction management
* Job processing statistics
* Job restart
* Skip and Resource management

You can also scale spring batch applications using its portioning techniques.

**Features of Spring Batch**

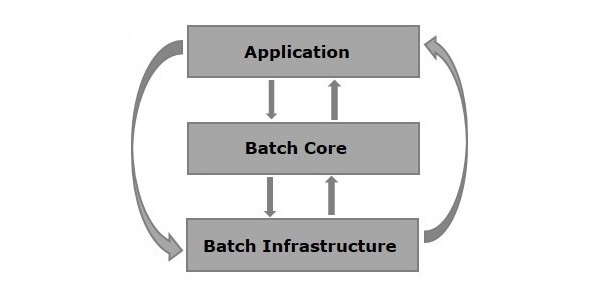
Following are the notable features of Spring Batch −

* **Flexibility** − Spring Batch applications are flexible. You simply need to change an XML file to alter the order of processing in an application.
* **Maintainability** − Spring Batch applications are easy to maintain. A Spring Batch job includes steps and each step can be decoupled, tested, and updated, without effecting the other steps.
* **Scalability** − Using the portioning techniques, you can scale the Spring Batch applications. These techniques allow you to −
  + Execute the steps of a job in parallel.
  + Execute a single thread in parallel.
* **Reliability** − In case of any failure, you can restart the job from exactly where it was stopped, by decoupling the steps.
* **Support for multiple file formats** − Spring Batch provides support for a large set of readers and writers such as XML, Flat file, CSV, MYSQL, Hibernate, JDBC, Mongo, Neo4j, etc.
* **Multiple ways to launch a job** − You can launch a Spring Batch job using web applications, Java programs, Command Line, etc.

In addition to these, Spring Batch applications support −

* Automatic retry after failure.
* Tracking status and statistics during the batch execution and after completing the batch processing.
* To run concurrent jobs.
* Services such as logging, resource management, skip, and restarting the processing.
* **Spring Batch – Architecture**

Following is the diagrammatic representation of the architecture of Spring Batch. As depicted in the figure, the architecture contains three main components namely, **Application, Batch Core**, and **Batch Infrastructure**.



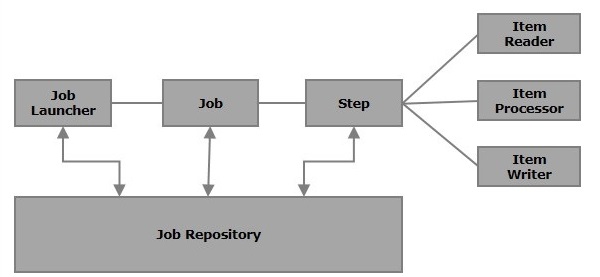
**Application** − This component contains all the jobs and the code we write using the Spring Batch framework.

**Batch Core** − This component contains all the API classes that are needed to control and launch a Batch Job.

**Batch Infrastructure** − This component contains the readers, writers, and services used by both application and Batch core components.

## Components of Spring Batch:

The following illustration shows the different components of Spring Batch and how they are connected with each other.



### **Job**

In a Spring Batch application, a job is the batch process that is to be executed. It runs from start to finish without interruption. This job is further divided into steps (or a job contains steps).

We will configure a job in Spring Batch using an XML file or a Java class. Following is the XML configuration of a Job in Spring Batch.

<job id = "jobid">

<step id = "step1" next = "step2"/>

<step id = "step2" next = "step3"/>

<step id = "step3"/>

</job>

A Batch job is configured within the tags <job></job>. It has an attribute named **id**. Within these tags, we define the definition and ordering of the steps.

**Restartable** − In general, when a job is running and we try to start it again that is considered as **restart** and it will be started again. To avoid this, you need to set the **restartable** value to **false** as shown below.

<job id = "jobid" restartable = "false" >

</job>

### **Step**

A **step** is an independent part of a job which contains the necessary information to define and execute the job (its part).

As specified in the diagram, each step is composed of an ItemReader, ItemProcessor (optional) and an ItemWriter. **A job may contain one or more steps**.

### **Readers, Writers, and Processors**

An **item reader** reads data into a Spring Batch application from a particular source, whereas an **item writer** writes data from the Spring Batch application to a particular destination.

An **Item processor** is a class which contains the processing code which processes the data read into the spring batch. If the application reads **"n"**records, then the code in the processor will be executed on each record.

When no reader and writer are given, a **tasklet** acts as a processor for SpringBatch. It processes only a single task. For example, if we are writing a job with a simple step in it where we read data from MySQL database and process it and write it to a file (flat), then our step uses −

* A **reader** which reads from MySQL database.
* A **writer** which writes to a flat file.
* A **custom processor** which processes the data as per our wish.

<job id = "helloWorldJob">

<step id = "step1">

<tasklet>

<chunk reader = "mysqlReader" writer = "fileWriter"

processor = "CustomitemProcessor" ></chunk>

</tasklet>

</step>

</ job>

Spring Batch provides a long list of **readers** and **writers**. Using these predefined classes, we can define beans for them.

* **JobRepository**

A Job repository in Spring Batch provides Create, Retrieve, Update, and Delete (CRUD) operations for the JobLauncher, Job, and Step implementations. We will define a job repository in an XML file as shown below.

<job-repository id = "jobRepository"/>

In addition to **id**, there are some more options (optional) available. Following is the configuration of job repository with all the options and their default values.

<job-repository id = "jobRepository"

data-source = "dataSource"

transaction-manager = "transactionManager"

isolation-level-for-create = "SERIALIZABLE"

table-prefix = "BATCH\_"

max-varchar-length = "1000"/>

**In-Memory Repository** − In case you don’t want to persist the domain objects of the Spring Batch in the database, you can configure the in-memory version of the jobRepository as shown below.

<bean id = "jobRepository"

class = "org.springframework.batch.core.repository.support.MapJobRepositoryFactoryBean ">

<property name = "transactionManager" ref = "transactionManager"/>

</bean>

### **JobLauncher**

JobLauncher is an interface which launces the Spring Batch job with the **given set of parameters**. **SimpleJoblauncher** is the class which implements the **JobLauncher** interface. Following is the configuration of the JobLauncher.

<bean id = "jobLauncher"

class = "org.springframework.batch.core.launch.support.SimpleJobLauncher">

<property name = "jobRepository" ref = "jobRepository" />

</bean>

### **JobInstance**

A **JobIinstance** represents the logical run of a job; it is created when we run a job. Each job instance is differentiated by the name of the job and the parameters passed to it while running.

If a JobInstance execution fails, the same JobInstance can be executed again. Hence, each JobInstance can have multiple job executions.

### **JobExecution and StepExecution**

JobExecution and StepExecution are the representation of the execution of a job/step. They contain the run information of the job/step such as start time (of job/step), end time (of job/step).

* **Spring Batch - Configuration**

While writing a Spring Batch application, we will configure the job, step, JobLauncher, JobRepository, Transaction Manager, readers, and writers using the XML tags provided in the Spring Batch namespace. Therefore, you need to include this namespace in your XML file as shown below.

<beans xmlns = "http://www.springframework.org/schema/beans"

xmlns:batch = "http://www.springframework.org/schema/batch"

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

xsi:schemaLocation = "http://www.springframework.org/schema/batch

http://www.springframework.org/schema/batch/spring-batch-2.2.xsd

http://www.springframework.org/schema/bean

http://www.springframework.org/schema/beans/spring-beans-3.2.xsd">

In the following sections, we will discuss the various tags, their attributes and examples, available in the Spring Batch namespace.

* **Job**

This tag is used to define/configure the job of the SpringBatch. It contains a set of steps and it can be launched using the JobLauncher.

This tag has 2 attributes as listed below −

|  |  |
| --- | --- |
| **S.No** | **Attribute & Description** |
| 1 | **Id**  It is the Id of the job, it is mandatory to specify value to this attribute. |
| 2 | **restartable**  This is the attribute which is used to specify whether the job is restartable or not. This attribute is optional. |

Following is the XML configuration of the job of a SpringBatch.

<job id = "jobid" restartable = "false" >

. . . . . . . .

. . . . . . . .

. . . . . . . . // Step definitions

</job>

* **Step**

This tag is used to define/configure the steps of a SpringBatch job. It has the following three attributes −

|  |  |
| --- | --- |
| **S.No** | **Attribute & Description** |
| 1 | **Id**  It is the Id of the job, it is mandatory to specify value to this attribute. |
| 2 | **next**  It is the shortcut to specify the next step. |
| 3 | **parent**  It is used to specify the name of the parent bean from which the configuration should inherit. |

Following is the XML configuration of the step of a SpringBatch.

<job id = "jobid">

<step id = "step1" next = "step2"/>

<step id = "step2" next = "step3"/>

<step id = "step3"/>

</job>

* **Chunk**

This tag is used to define/configure a chunk of a **tasklet**. It has the following four attributes –

|  |  |
| --- | --- |
| **S.No** | **Attribute & Description** |
| 1 | **reader**  It represents the name of the item reader bean. It accepts the value of the type **org.springframework.batch.item.ItemReader**. |
| 2 | **writer**  It represents the name of the item reader bean. It accepts the value of the type **org.springframework.batch.item.ItemWriter**. |
| 3 | **processor**  It represents the name of the item reader bean. It accepts the value of the type **org.springframework.batch.item.ItemProcessor**. |
| 4 | **commit-interval**  It is used to specify the number of items to be processed before committing the transaction. |

Following is the XML configuration of the chunk of a SpringBatch.

<batch:step id = "step1">

<batch:tasklet>

<batch:chunk reader = "xmlItemReader"

writer = "mysqlItemWriter" processor = "itemProcessor" commit-interval = "10">

</batch:chunk>

</batch:tasklet>

</batch:step>

* **JobRepository**

The JobRepository Bean is used to configure the JobRepository using a relational database. This bean is associated with the class of type **org.springframework.batch.core.repository.JobRepository**.

|  |  |
| --- | --- |
| **S.No** | **Attribute & Description** |
| 1 | **dataSource**  It is used to specify the bean name which defines the datasource. |
| 2 | **transactionManager**  It is used specify the name of the bean which defines the transactionmanager. |
| 3 | **databaseType**  It specifies the type of the relational database used in the job repository. |

Following is the example configuration of the JobRepository.

<bean id = "jobRepository"

class = "org.springframework.batch.core.repository.support.JobRepositoryFactoryBean">

<property name = "dataSource" ref = "dataSource" />

<property name = "transactionManager" ref="transactionManager" />

<property name = "databaseType" value = "mysql" />

</bean>

* **JobLauncher**

The JobLauncher bean is used to configure the JobLauncher. It is associated with the class **org.springframework.batch.core.launch.support.SimpleJobLauncher**(in our programs). This bean has one property named **jobrepository**, and it is used to specify the name of the bean which defines the **jobrepository**.

Following is the example configuration of the jobLauncher.

<bean id = "jobLauncher"

class = "org.springframework.batch.core.launch.support.SimpleJobLauncher">

<property name = "jobRepository" ref = "jobRepository" />

</bean>

* **TransactionManager**

The TransactionManager bean is used to configure the TransactionManager using a relational database. This bean is associated with the class of type **org.springframework.transaction.platform.TransactionManager**

<bean id = "transactionManager" class = "org.springframework.batch.support.transaction.ResourcelessTransactionManager" />

* **DataSource**

The datasource bean is used to configure the **Datasource**. This bean is associated with the class of type **org.springframework.jdbc.datasource.DriverManagerDataSource**.

|  |  |
| --- | --- |
| **S.No** | **Attribute & Description** |
| 1 | **driverClassName**  This specifies the class name of the driver used to connect with the database. |
| 2 | **url**  This specifies the URL of the database. |
| 3 | **username**  This specifies the username to connect with the database. |
| 4 | **password**  This specifies the password to connect with the database. |

Following is the example configuration of the **datasource**.

<bean id = "dataSource"

class = "org.springframework.jdbc.datasource.DriverManagerDataSource">

<property name = "driverClassName" value = "com.mysql.jdbc.Driver" />

<property name = "url" value = "jdbc:mysql://localhost:3306/springbatch" />

<property name = "username" value = "root" />

<property name = "password" value = "123456" />

</bean>

* **Spring Batch - Readers, Writers & Processors**

An **Item Reader** reads data into the spring batch application from a particular source, whereas an **Item Writer** writes data from Spring Batch application to a particular destination.

An **Item processor** is a class which contains the processing code which processes the data read in to the spring batch. If the application reads n records the code in the processor will be executed on each record.

A **chunk** is a child element of the **tasklet**. It is used to perform read, write, and processing operations. We can configure reader, writer, and processors using this element, within a step as shown below.

<batch:job id = "helloWorldJob">

<batch:step id = "step1">

<batch:tasklet>

<batch:chunk reader = "cvsFileItemReader" writer = "xmlItemWriter"

processor = "itemProcessor" commit-interval = "10">

</batch:chunk>

</batch:tasklet>

</batch:step>

</batch:job>

Spring Batch provides readers and writers to read and write data form various file systems/databases such as MongoDB, Neo4j, MySQL, XML, flatfile, CSV, etc.

To include a reader in your application, you need to define a bean for that reader, provide values to all the required properties within the bean, and pass the **id** of such bean as a value to the attribute of the chunk element **reader**(same for **writer**).

## ItemReader

It is the entity of a step (of a batch process) which reads data. An ItemReader reads one item a time. Spring Batch provides an Interface **ItemReader**. All the **readers** implement this interface.

Following are some of the predefined ItemReader classes provided by Spring Batch to read from various sources.

|  |  |
| --- | --- |
| **Reader** | **Purpose** |
| FlatFIleItemReader | To read data from flat files. |
| StaxEventItemReader | To read data from XML files. |
| StoredProcedureItemReader | To read data from the stored procedures of a database. |
| JDBCPagingItemReader | To read data from relational databases database. |
| MongoItemReader | To read data from MongoDB. |
| Neo4jItemReader | To read data from Neo4jItemReader. |

We need to configure the **ItemReaders** by creating the beans. Following is an example of **StaxEventItemReader** which reads data from an XML file.

<bean id = "mysqlItemWriter"

class = "org.springframework.batch.item.xml.StaxEventItemWriter">

<property name = "resource" value = "file:xml/outputs/userss.xml" />

<property name = "marshaller" ref = "reportMarshaller" />

<property name = "rootTagName" value = "Tutorial" />

</bean>

<bean id = "reportMarshaller"

class = "org.springframework.oxm.jaxb.Jaxb2Marshaller">

<property name = "classesToBeBound">

<list>

<value>Tutorial</value>

</list>

</property>

</bean>

As observed, while configuring, we need to specify the respective class name of the required reader and we need to provide values to all the required properties.

## ItemWriter

It is the element of the **step** of a batch process which writes data. An ItemWriter writes one item a time. Spring Batch provides an Interface **ItemWriter**. All the writers implement this interface.

Following are some of the predefined ItemWriter classes provided by Spring Batch to read from various sources.

|  |  |
| --- | --- |
| **Writer** | **Purpose** |
| FlatFIleItemWriter | To write data into flat files. |
| StaxEventItemWriter | To write data into XML files. |
| StoredProcedureItemWriter | To write data into the stored procedures of a database. |
| JDBCPagingItemWriter | To write data into relational databases database. |
| MongoItemWriter | To write data into MongoDB. |
| Neo4jItemWriter | To write data into Neo4j. |

In same way, we need to configure the ItemWriters by creating the beans. Following is an example of **JdbcCursorItemReader** which writes data to an MySQL database.

<bean id = "dbItemReader" class = "org.springframework.batch.item.database.JdbcCursorItemReader" scope = "step">

<property name = "dataSource" ref = "dataSource" />

<property name = "sql" value = "select \* from tutorialsdata" />

<property name = "rowMapper">

<bean class = "TutorialRowMapper" />

</property>

</bean>

## Item Processor

**ItemProcessor**: An ItemProcessor is used to process the data. When the given item is not valid it returns **null**, else it processes the given item and returns the processed result. The interface **ItemProcessor<I,O>** represents the processor.

**Tasklet class** − When no **reader** and **writer** are given, a Tasklet acts as a processor for SpringBatch. It processes only single task.

We can define a custom item processor by implementing the interface **ItemProcessor** of the package **org.springframework.batch.item.ItemProcessor**. This ItemProcessor class accepts an object and processes the data and returns the processed data as another object.

In a batch process, if **"n"** records or data elements are read, then for each record, it will read the data, process it, and writes the data in the writer. To process the data, it relays on the processor passed.

For example, let’s suppose you have written code to load a particular PDF document, create a new page, write the data item on to the PDF in a tabular format. If you execute this application, it reads all the data items from the XML document, stores them in the MySQL database, and prints them in the given PDF document in individual pages.

### **Example**

Following is a sample ItemProcessor class.

import org.springframework.batch.item.ItemProcessor;

public class CustomItemProcessor implements ItemProcessor<Tutorial, Tutorial> {

@Override

public Tutorial process(Tutorial item) throws Exception {

System.out.println("Processing..." + item);

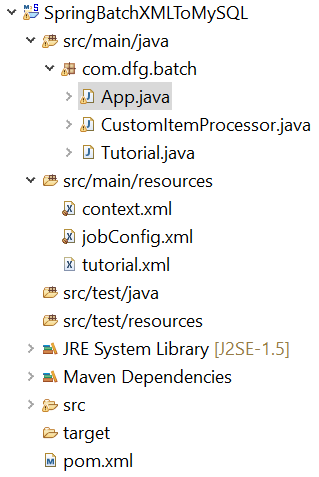
return item;

}

}

# **Spring Batch - XML to MySQL**

* **Project Structure :**



* **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.dfg.batch</groupId>

<artifactId>SpringBatchXMLToMySQL</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBatchXMLToMySQL</name>

<properties>

<jdk.version>1.6</jdk.version>

<spring.version>4.0.0.RELEASE</spring.version>

<spring.batch.version>3.0.7.RELEASE</spring.batch.version>

<mysql.driver.version>5.1.25</mysql.driver.version>

<junit.version>4.11</junit.version>

</properties>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring jdbc, for database -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring XML to/back object -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- MySQL database driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.driver.version}</version>

</dependency>

<!-- Spring Batch dependencies -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-core</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-infrastructure</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<!-- Spring Batch unit test -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-test</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.apache.pdfbox</groupId>

<artifactId>pdfbox</artifactId>

<version>2.0.8</version>

</dependency>

<!-- Junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<finalName>SpringBatchXMLToMySQL</finalName>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-eclipse-plugin</artifactId>

<version>2.9</version>

<configuration>

<downloadSources>true</downloadSources>

<downloadJavadocs>false</downloadJavadocs>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Reader** − The reader we are using in the application is **StaxEventItemReader** to read data from XML documents.

Following is the input XML document we are using in this application. This document holds data records which specify details like tutorial id, tutorial author, tutorial title, submission date, tutorial icon, and tutorial description.

<?xml version="1.0" encoding="UTF-8"?>

<tutorials>

<tutorial>

<tutorial\_id>1001</tutorial\_id>

<tutorial\_author>Sanjay</tutorial\_author>

<tutorial\_title>Learn Java</tutorial\_title>

<submission\_date>06-05-2007</submission\_date>

<tutorial\_icon>https://www.tutorialspoint.com/java/images/java-minilogo.jpg</tutorial\_icon>

<tutorial\_description>Java is a high-level programming language originally

developed by Sun Microsystems and released in 1995.

Java runs on a variety of platforms.

This tutorial gives a complete understanding of Java.');</tutorial\_description>

</tutorial>

<tutorial>

<tutorial\_id>1002</tutorial\_id>

<tutorial\_author>Abdul S</tutorial\_author>

<tutorial\_title>Learn MySQL</tutorial\_title>

<submission\_date>19-04-2007</submission\_date>

<tutorial\_icon>https://www.tutorialspoint.com/mysql/images/mysql-minilogo.jpg</tutorial\_icon>

<tutorial\_description>MySQL is the most popular

Open Source Relational SQL database management system.

MySQL is one of the best RDBMS being used for developing web-based software applications.

This tutorial will give you quick start with MySQL

and make you comfortable with MySQL programming.</tutorial\_description>

</tutorial>

<tutorial>

<tutorial\_id>1003</tutorial\_id>

<tutorial\_author>Krishna Kasyap</tutorial\_author>

<tutorial\_title>Learn JavaFX</tutorial\_title>

<submission\_date>06-07-2017</submission\_date>

<tutorial\_icon>https://www.tutorialspoint.com/javafx/images/javafx-minilogo.jpg</tutorial\_icon>

<tutorial\_description>JavaFX is a Java library used to build Rich Internet Applications.

The applications developed using JavaFX can run on various devices

such as Desktop Computers, Mobile Phones, TVs, Tablets, etc.

This tutorial, discusses all the necessary elements of JavaFX that are required

to develop effective Rich Internet Applications</tutorial\_description>

</tutorial>

</tutorials>

**Writer** −The **writer** we are using in the application is **JdbcBatchItemWriter** to write the data to MySQL database. Assume we have created a table in MySQL inside a table called **"** TUTORIALS **"**.

CREATE TABLE TUTORIALS(

tutorial\_id int(10) NOT NULL,

tutorial\_author VARCHAR(20),

tutorial\_title VARCHAR(50),

submission\_date VARCHAR(20),

tutorial\_icon VARCHAR(200),

tutorial\_description VARCHAR(1000)

);

**Processor** − The processor we are using in the application is a custom processor which writes the data of each record on the PDF document.

In batch process, if **"n"** records or data elements were read, then for each record, it will read the data, process it, and write the data in the Writer. To process the data, it relays on the processor passed. In this case, in the custom processor class, we have written code to load a particular PDF document, create a new page, write the data item onto the PDF in a tabular format.

Finally, if you execute this application, it reads all the data items from the XML document, stores them in the MySQL database, and prints them in the given PDF document in individual pages.

* **jobConfig.xml:**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:batch = *"http://www.springframework.org/schema/batch"*

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

xmlns:util = *"http://www.springframework.org/schema/util"*

xsi:schemaLocation = *"http://www.springframework.org/schema/batch*

*http://www.springframework.org/schema/batch/spring-batch-2.2.xsd*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/util*

*http://www.springframework.org/schema/util/spring-util-3.0.xsd "*>

<import resource = *"context.xml"* />

<!-- Create Spring Batch Job -->

<batch:job id = *"xmltomysqljob"*>

<batch:step id = *"step1"*>

<batch:tasklet>

<batch:chunk reader = *"xmlItemReader"* writer = *"mysqlItemWriter"* processor = *"itemProcessor"* commit-interval=*"10"*>

</batch:chunk>

</batch:tasklet>

</batch:step>

</batch:job>

<!-- Item Reader -->

<bean id = *"xmlItemReader"*

class = *"org.springframework.batch.item.xml.StaxEventItemReader"*>

<property name = *"fragmentRootElementName"* value = *"tutorial"* />

<property name = *"resource"* value = *"classpath:tutorial.xml"* />

<property name = *"unmarshaller"* ref = *"customUnMarshaller"* />

</bean>

<bean id = *"customUnMarshaller"* class = *"org.springframework.oxm.xstream.XStreamMarshaller"*>

<property name = *"aliases"*>

<util:map id = *"aliases"*>

<entry key = *"tutorial"* value = *"com.dfg.batch.Tutorial"* />

</util:map>

</property>

</bean>

<!-- Item Writer -->

<bean id = *"mysqlItemWriter"* class = *"org.springframework.batch.item.database.JdbcBatchItemWriter"*>

<property name = *"dataSource"* ref = *"dataSource"* />

<property name = *"sql"*>

<value>

<![CDATA[insert into tutorials (tutorial\_id, tutorial\_author, tutorial\_title,

submission\_date, tutorial\_icon, tutorial\_description)

values (:tutorial\_id, :tutorial\_author, :tutorial\_title, :submission\_date,

:tutorial\_icon, :tutorial\_description);]]>

</value>

</property>

<property name = *"itemSqlParameterSourceProvider"*>

<bean class = *"org.springframework.batch.item.database.BeanPropertyItemSqlParameterSourceProvider"* />

</property>

</bean>

<!-- Item Processor -->

<bean id = *"itemProcessor"* class = *"com.dfg.batch.CustomItemProcessor"* />

</beans>

* **Context.xml:**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:jdbc = *"http://www.springframework.org/schema/jdbc"*

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

xsi:schemaLocation = *"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/jdbc*

*http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd"*>

<!-- Create JobRepository -->

<bean id = *"jobRepository"*

class = *"org.springframework.batch.core.repository.support.JobRepositoryFactoryBean"*>

<property name = *"dataSource"* ref = *"dataSource"* /> <!-- Inject DataSource -->

<property name = *"transactionManager"* ref = *"transactionManager"* /> <!-- Inject Txn Manager -->

<property name = *"databaseType"* value = *"mysql"* />

</bean>

<!-- Create Txn Manager -->

<bean id = *"transactionManager"*

class = *"org.springframework.batch.support.transaction.ResourcelessTransactionManager"* />

<!-- Create DataSource -->

<bean id = *"dataSource"*

class = *"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name = *"driverClassName"* value = *"com.mysql.jdbc.Driver"* />

<property name = *"url"* value = *"jdbc:mysql://localhost:3306/springorm"* />

<property name = *"username"* value = *"root"* />

<property name = *"password"* value = *"123456"* />

</bean>

<!-- Create Job Launcher -->

<bean id = *"jobLauncher"*

class = *"org.springframework.batch.core.launch.support.SimpleJobLauncher"*>

<property name = *"jobRepository"* ref = *"jobRepository"* />

</bean>

<!-- create job-meta tables automatically -->

<jdbc:initialize-database data-source = *"dataSource"*>

<jdbc:script location = *"org/springframework/batch/core/schema-drop-mysql.sql"*/>

<jdbc:script location = *"org/springframework/batch/core/schema-mysql.sql"*/>

</jdbc:initialize-database>

</beans>

* **Tutorial.java:**

**package** com.dfg.batch;

**public** **class** Tutorial {

**private** **int** tutorial\_id;

**private** String tutorial\_author;

**private** String tutorial\_title;

**private** String submission\_date;

**private** String tutorial\_icon;

**private** String tutorial\_description;

@Override

**public** String toString() {

**return** " [id=" + tutorial\_id + ", author=" + tutorial\_author

+ ", title=" + tutorial\_title + ", date=" + submission\_date + ", icon ="

+tutorial\_icon +", description = "+tutorial\_description+"]";

}

**public** **int** getTutorial\_id() {

**return** tutorial\_id;

}

**public** **void** setTutorial\_id(**int** tutorial\_id) {

**this**.tutorial\_id = tutorial\_id;

}

**public** String getTutorial\_author() {

**return** tutorial\_author;

}

**public** **void** setTutorial\_author(String tutorial\_author) {

**this**.tutorial\_author = tutorial\_author;

}

**public** String getTutorial\_title() {

**return** tutorial\_title;

}

**public** **void** setTutorial\_title(String tutorial\_title) {

**this**.tutorial\_title = tutorial\_title;

}

**public** String getSubmission\_date() {

**return** submission\_date;

}

**public** **void** setSubmission\_date(String submission\_date) {

**this**.submission\_date = submission\_date;

}

**public** String getTutorial\_icon() {

**return** tutorial\_icon;

}

**public** **void** setTutorial\_icon(String tutorial\_icon) {

**this**.tutorial\_icon = tutorial\_icon;

}

**public** String getTutorial\_description() {

**return** tutorial\_description;

}

**public** **void** setTutorial\_description(String tutorial\_description) {

**this**.tutorial\_description = tutorial\_description;

}

}

* **CustomItemProcessor.java:**

**package** com.dfg.batch;

**import** java.io.File;

**import** java.io.IOException;

**import** org.apache.pdfbox.pdmodel.PDDocument;

**import** org.apache.pdfbox.pdmodel.PDPage;

**import** org.apache.pdfbox.pdmodel.PDPageContentStream;

**import** org.apache.pdfbox.pdmodel.font.PDType1Font;

**import** org.springframework.batch.item.ItemProcessor;

//This class acts as Item Processor

//It reads the data from XML file creates a PDF Page and writes the data into it in the form of Table

**public** **class** CustomItemProcessor **implements** ItemProcessor<Tutorial, Tutorial> {

**public** **static** **void** drawTable(PDPage page, PDPageContentStream contentStream,

**float** y, **float** margin, String[][] content) **throws** IOException {

**final** **int** rows = content.length;

**final** **int** cols = content[0].length;

**final** **float** rowHeight = 50;

**final** **float** tableWidth = page.getMediaBox().getWidth()-(2\*margin);

**final** **float** tableHeight = rowHeight \* rows;

**final** **float** colWidth = tableWidth/(**float**)cols;

**final** **float** cellMargin=5f;

// draw the rows

**float** nexty = y ;

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

contentStream.~~drawLine~~(margin,nexty,margin+tableWidth,nexty);

nexty-= rowHeight;

}

//draw the columns

**float** nextx = margin;

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

contentStream.~~drawLine~~(nextx,y,nextx,y-tableHeight);

nextx += colWidth;

}

// now add the text

contentStream.setFont(PDType1Font.***HELVETICA\_BOLD***,12);

**float** textx = margin+cellMargin;

**float** texty = y-15;

**for**(**int** i = 0; i < content.length; i++){

**for**(**int** j = 0 ; j < content[i].length; j++){

String text = content[i][j];

contentStream.beginText();

contentStream.~~moveTextPositionByAmount~~(textx,texty);

contentStream.~~drawString~~(text);

contentStream.endText();

textx += colWidth;

}

texty-=rowHeight;

textx = margin+cellMargin;

}

}

**public** Tutorial process(Tutorial item) **throws** Exception {

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

// Creating PDF document object

PDDocument doc = PDDocument.*load*(**new** File("C:\\Users\\ambhaler\\138947\\SpringWorkspace\\SpringBatch\\SpringBatchTest\\test.pdf"));

// Creating a blank page

PDPage page = **new** PDPage();

doc.addPage( page );

PDPageContentStream contentStream = **new** PDPageContentStream(doc, page);

String[][] content = {{"Id",""+item.getTutorial\_id()},

{"Title", item.getTutorial\_title()},

{"Authour", item.getTutorial\_author()},

{"Submission Date", item.getSubmission\_date()}} ;

*drawTable*(page, contentStream, 700, 100, content);

contentStream.close();

doc.save("C:\\Users\\ambhaler\\138947\\SpringWorkspace\\SpringBatch\\SpringBatchTest\\test.pdf" );

**return** item;

}

}

* **App.java:**

**package** com.dfg.batch;

**import** org.springframework.batch.core.Job;

**import** org.springframework.batch.core.JobExecution;

**import** org.springframework.batch.core.JobParameters;

**import** org.springframework.batch.core.launch.JobLauncher;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** App {

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

String springConfig = "jobConfig.xml";

// Creating the application context object

ApplicationContext context = **new** ClassPathXmlApplicationContext(springConfig);

// Creating the job launcher

JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

// Creating the job

Job job = (Job) context.getBean("xmltomysqljob");

// Executing the JOB

JobExecution execution = jobLauncher.run(job, **new** JobParameters());

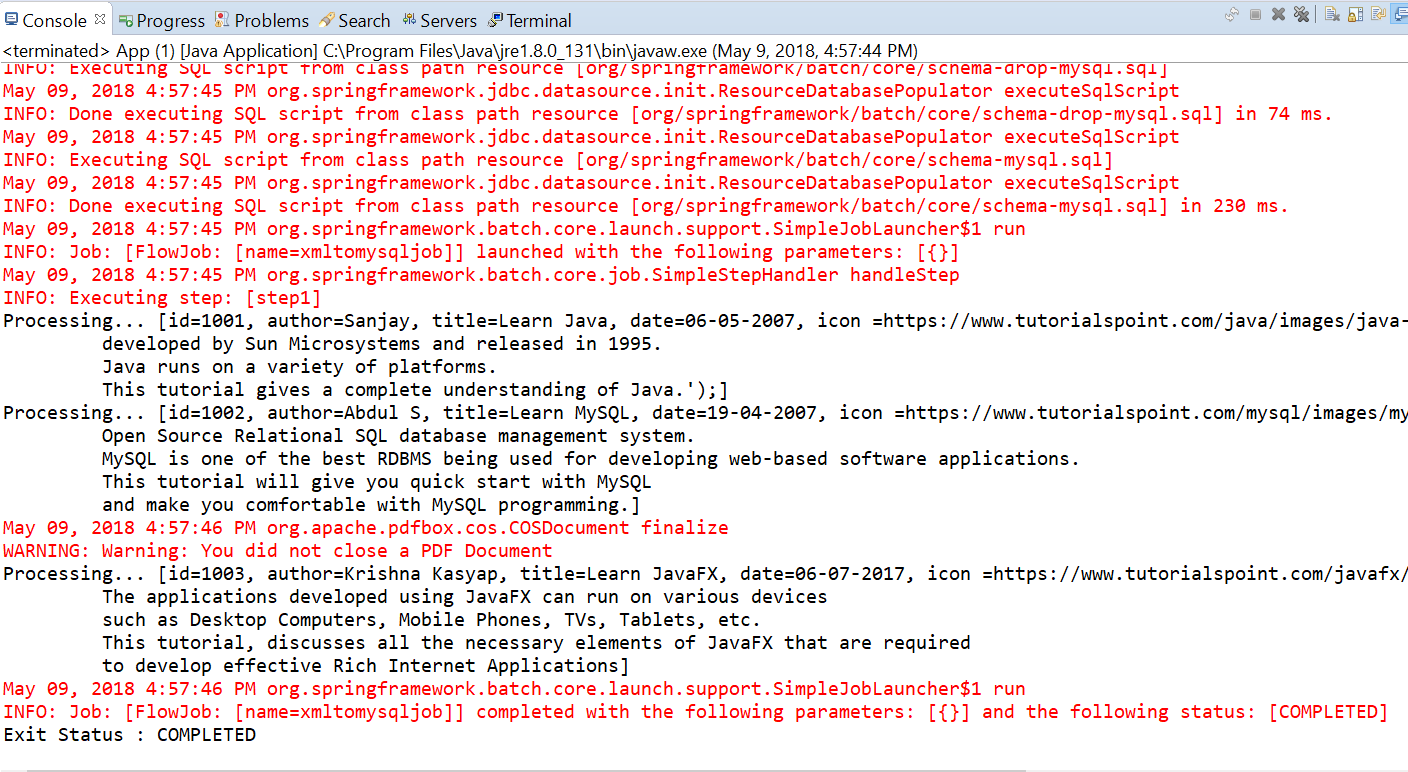
System.***out***.println("Exit Status : " + execution.getStatus());

}

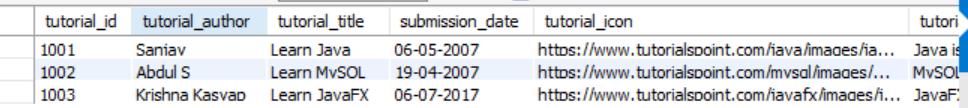
}

Execute App.java as Java Application

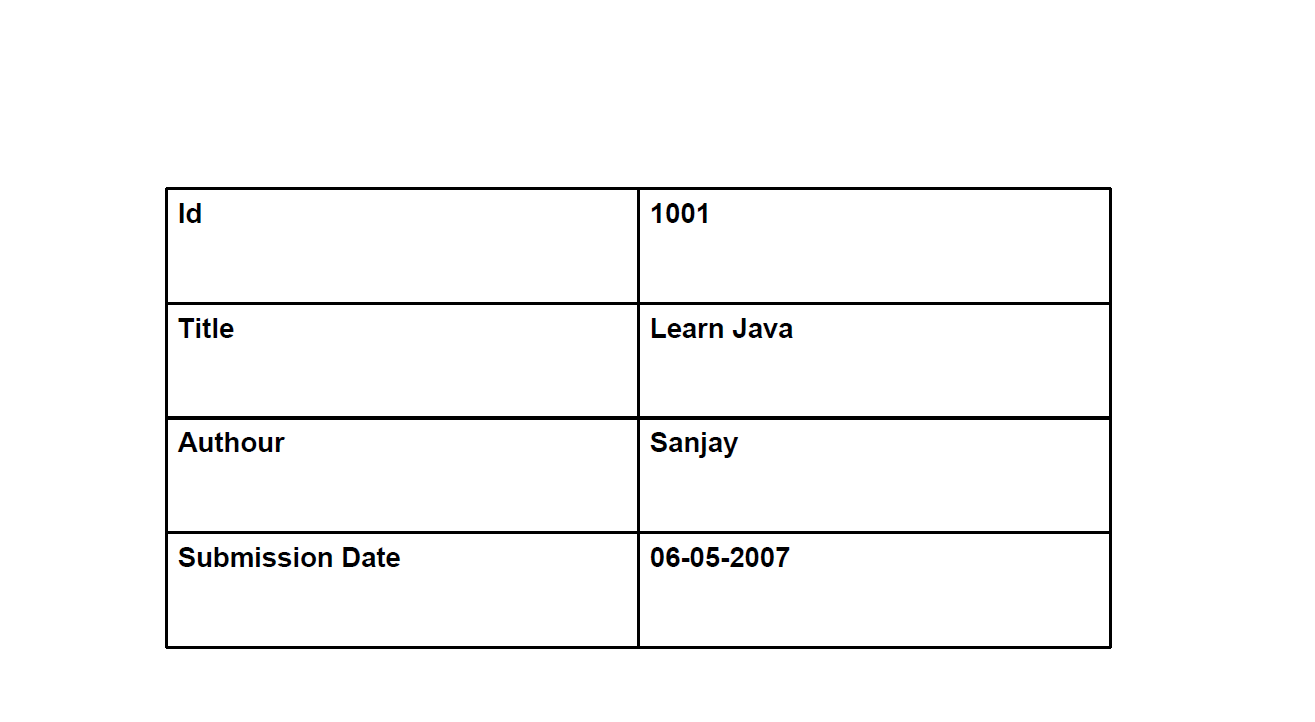
**Console :**



**Mysql :**

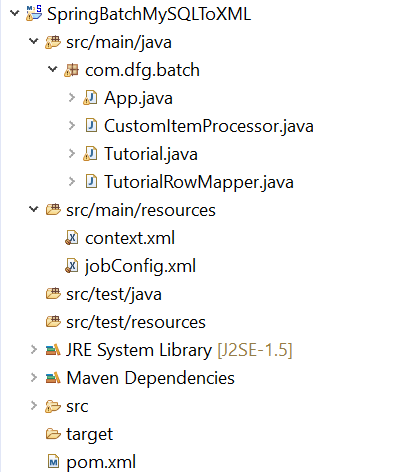


**PDF:**



# **Spring Batch - MySQL to XML**

* **Project Structure :**



* **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.dfg.batch</groupId>

<artifactId>SpringBatchXMLToMySQL</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBatchXMLToMySQL</name>

<properties>

<jdk.version>1.6</jdk.version>

<spring.version>4.0.0.RELEASE</spring.version>

<spring.batch.version>3.0.7.RELEASE</spring.batch.version>

<mysql.driver.version>5.1.25</mysql.driver.version>

<junit.version>4.11</junit.version>

</properties>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring jdbc, for database -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring XML to/back object -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- MySQL database driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.driver.version}</version>

</dependency>

<!-- Spring Batch dependencies -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-core</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-infrastructure</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<!-- Spring Batch unit test -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-test</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.apache.pdfbox</groupId>

<artifactId>pdfbox</artifactId>

<version>2.0.8</version>

</dependency>

<!-- Junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<finalName>SpringBatchXMLToMySQL</finalName>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-eclipse-plugin</artifactId>

<version>2.9</version>

<configuration>

<downloadSources>true</downloadSources>

<downloadJavadocs>false</downloadJavadocs>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Reader** − The reader we are using in the application is **JdbcCursorItemReader** to read data from MySQL database.

Assume we have created a table in the MySQL database as shown below −

CREATE TABLE `tutorials` (

`tutorial\_id` int(10) NOT NULL,

`tutorial\_author` varchar(20) DEFAULT NULL,

`tutorial\_title` varchar(50) DEFAULT NULL,

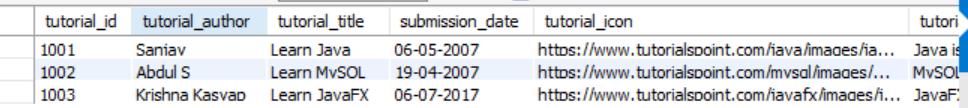
`submission\_date` varchar(20) DEFAULT NULL,

`tutorial\_icon` varchar(200) DEFAULT NULL,

`tutorial\_description` varchar(1000) DEFAULT NULL

) ;

Assume we have inserted the following records in to it.



**Writer** − The Writer we are using in the application is **StaxEventItemWriter**to write the data to the XML file.

**Processor** − The Processor we are using in the application is a custom processor which just prints the records read from the MySQL.

* **jobConfig.xml:**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:batch = *"http://www.springframework.org/schema/batch"*

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

xmlns:util = *"http://www.springframework.org/schema/util"*

xsi:schemaLocation = *"http://www.springframework.org/schema/batch*

*http://www.springframework.org/schema/batch/spring-batch-2.2.xsd*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/util*

*http://www.springframework.org/schema/util/spring-util-3.0.xsd "*>

<import resource = *"context.xml"* />

<!-- Create Spring Batch Job -->

<batch:job id = *"mysqltoxmljob"*>

<batch:step id = *"step1"*>

<batch:tasklet>

<batch:chunk reader = *"dbItemReader"*

writer = *"mysqlItemWriter"* processor = *"itemProcessor"* commit-interval = *"10"*>

</batch:chunk>

</batch:tasklet>

</batch:step>

</batch:job>

<bean id = *"dbItemReader"*

class = *"org.springframework.batch.item.database.JdbcCursorItemReader"* scope = *"step"*>

<property name = *"dataSource"* ref = *"dataSource"* />

<property name = *"sql"* value = *"select \* from tutorials"* />

<property name = *"rowMapper"*>

<bean class = *"com.dfg.batch.TutorialRowMapper"* />

</property>

</bean>

<bean id = *"mysqlItemWriter"*

class = *"org.springframework.batch.item.xml.StaxEventItemWriter"*>

<property name = *"resource"* value = *"file:tutorials.xml"* />

<property name = *"marshaller"* ref = *"reportMarshaller"* />

<property name = *"rootTagName"* value = *"Tutorial"* />

</bean>

<bean id = *"reportMarshaller"* class = *"org.springframework.oxm.jaxb.Jaxb2Marshaller"*>

<property name = *"classesToBeBound"*>

<list>

<value>com.dfg.batch.Tutorial</value>

</list>

</property>

</bean>

<bean id = *"report"* class = *"Report"* scope = *"prototype"* />

<bean id = *"itemProcessor"* class = *"com.dfg.batch.CustomItemProcessor"* />

</beans>

* **Context.xml:**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:batch = *"http://www.springframework.org/schema/batch"*

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

xmlns:util = *"http://www.springframework.org/schema/util"*

xsi:schemaLocation = *"http://www.springframework.org/schema/batch*

*http://www.springframework.org/schema/batch/spring-batch-2.2.xsd*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/util*

*http://www.springframework.org/schema/util/spring-util-3.0.xsd "*>

<import resource = *"context.xml"* />

<!-- Create Spring Batch Job -->

<batch:job id = *"mysqltoxmljob"*>

<batch:step id = *"step1"*>

<batch:tasklet>

<batch:chunk reader = *"dbItemReader"*

writer = *"mysqlItemWriter"* processor = *"itemProcessor"* commit-interval = *"10"*>

</batch:chunk>

</batch:tasklet>

</batch:step>

</batch:job>

<bean id = *"dbItemReader"*

class = *"org.springframework.batch.item.database.JdbcCursorItemReader"* scope = *"step"*>

<property name = *"dataSource"* ref = *"dataSource"* />

<property name = *"sql"* value = *"select \* from tutorials"* />

<property name = *"rowMapper"*>

<bean class = *"com.dfg.batch.TutorialRowMapper"* />

</property>

</bean>

<bean id = *"mysqlItemWriter"*

class = *"org.springframework.batch.item.xml.StaxEventItemWriter"*>

<property name = *"resource"* value = *"file:tutorials.xml"* />

<property name = *"marshaller"* ref = *"reportMarshaller"* />

<property name = *"rootTagName"* value = *"Tutorial"* />

</bean>

<bean id = *"reportMarshaller"* class = *"org.springframework.oxm.jaxb.Jaxb2Marshaller"*>

<property name = *"classesToBeBound"*>

<list>

<value>com.dfg.batch.Tutorial</value>

</list>

</property>

</bean>

<bean id = *"report"* class = *"Report"* scope = *"prototype"* />

<bean id = *"itemProcessor"* class = *"com.dfg.batch.CustomItemProcessor"* />

</beans>

* **Tutorial.java:**

**package** com.dfg.batch;

**import** javax.xml.bind.annotation.XmlAttribute;

**import** javax.xml.bind.annotation.XmlElement;

**import** javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "tutorials")

**public** **class** Tutorial {

**int** tutorial\_id;

String tutorial\_author;

String submission\_date;

String tutorial\_title;

@XmlAttribute(name = "tutorial\_id")

**public** **int** getTutorial\_id() {

**return** tutorial\_id;

}

**public** **void** setTutorial\_id(**int** tutorial\_id) {

**this**.tutorial\_id = tutorial\_id;

}

@XmlElement(name = "tutorial\_author")

**public** String getTutorial\_author() {

**return** tutorial\_author;

}

**public** **void** setTutorial\_author(String tutorial\_author) {

**this**.tutorial\_author = tutorial\_author;

}

@XmlElement(name = "tutorial\_title")

**public** String getTutorial\_title() {

**return** tutorial\_title;

}

**public** **void** setTutorial\_title(String tutorial\_title) {

**this**.tutorial\_title = tutorial\_title;

}

@XmlElement(name = "submission\_date")

**public** String getSubmission\_date() {

**return** submission\_date;

}

**public** **void** setSubmission\_date(String submission\_date) {

**this**.submission\_date = submission\_date;

}

**public** String toString() {

**return** " [Tutorial Id=" + tutorial\_id + ",Tutorial Author =" + tutorial\_author + ",Tutorial Title =" + tutorial\_title + ",Submission Date =" + submission\_date + "]";

}

}

* **CustomItemProcessor.java:**

**package** com.dfg.batch;

**import** org.springframework.batch.item.ItemProcessor;

//This class acts as Item Processor

**public** **class** CustomItemProcessor **implements** ItemProcessor<Tutorial, Tutorial> {

**public** Tutorial process(Tutorial item) **throws** Exception {

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

**return** item;

}

}

* **TutorialRowMapper.java:**

**package** com.dfg.batch;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** org.springframework.jdbc.core.RowMapper;

**public** **class** TutorialRowMapper **implements** RowMapper<Tutorial> {

**public** Tutorial mapRow(ResultSet rs, **int** rowNum) **throws** SQLException {

Tutorial tutorial = **new** Tutorial();

tutorial.setTutorial\_id(rs.getInt("tutorial\_id"));

tutorial.setTutorial\_author(rs.getString("tutorial\_author"));

tutorial.setTutorial\_title(rs.getString("tutorial\_title"));

tutorial.setSubmission\_date(rs.getString("submission\_date"));

**return** tutorial;

}

}

* **App.java:**

**package** com.dfg.batch;

**import** org.springframework.batch.core.Job;

**import** org.springframework.batch.core.JobExecution;

**import** org.springframework.batch.core.JobParameters;

**import** org.springframework.batch.core.launch.JobLauncher;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** App {

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

String springConfig = "jobConfig.xml";

// Creating the application context object

ApplicationContext context = **new** ClassPathXmlApplicationContext(springConfig);

// Creating the job launcher

JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

// Creating the job

Job job = (Job) context.getBean("mysqltoxmljob");

// Executing the JOB

JobExecution execution = jobLauncher.run(job, **new** JobParameters());

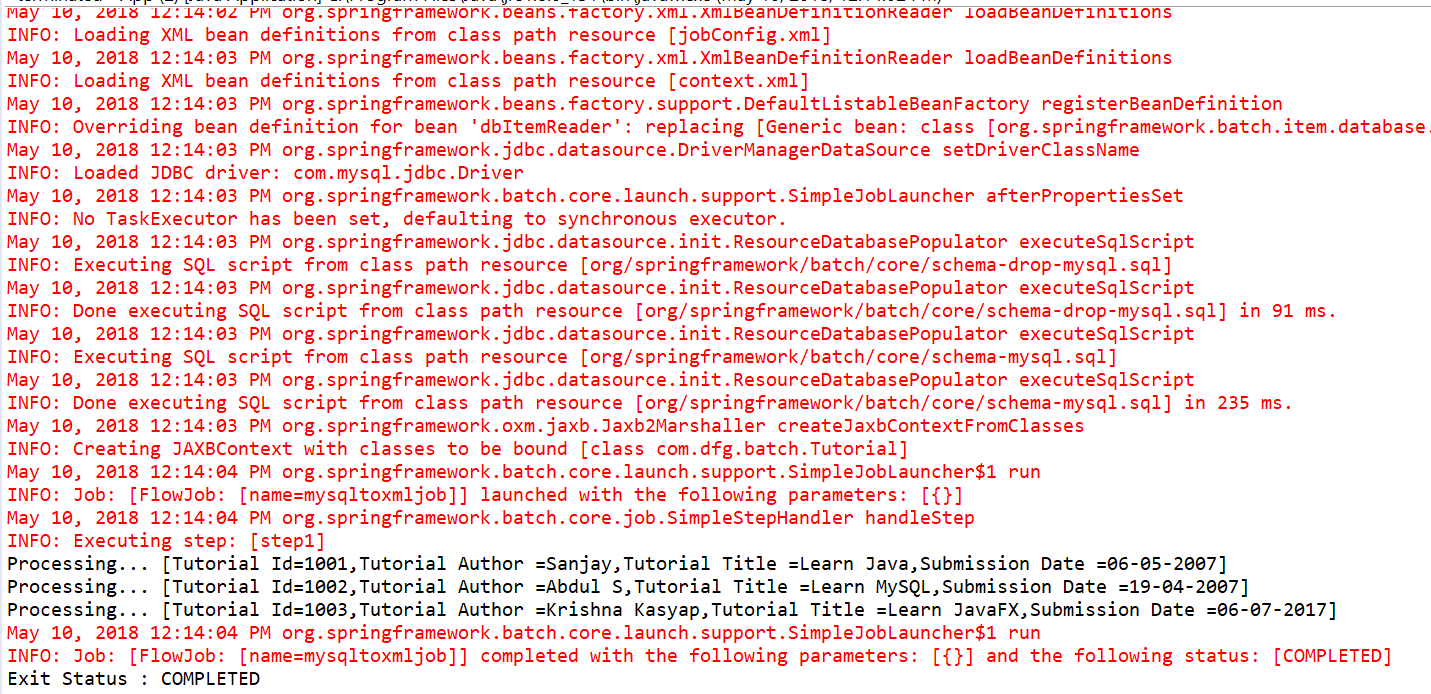
System.***out***.println("Exit Status : " + execution.getStatus());

}

}

Execute App.java as Java Application

**Console:**



**tutorials.xml:**

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

**<Tutorial>**

**<details tutorial\_id = "101">**

**<submission\_date>06-05-2007</submission\_date>**

**<tutorial\_author>Sanjay</tutorial\_author>**

**<tutorial\_title>Learn Java</tutorial\_title>**

**</details>**

**<details tutorial\_id = "102">**

**<submission\_date>19-04-2007</submission\_date>**

**<tutorial\_author>Abdul S</tutorial\_author>**

**<tutorial\_title>Learn MySQL</tutorial\_title>**

**</details>**

**<details tutorial\_id = "103">**

**<submission\_date>06-07-2017</submission\_date>**

**<tutorial\_author>Krishna Kasyap</tutorial\_author>**

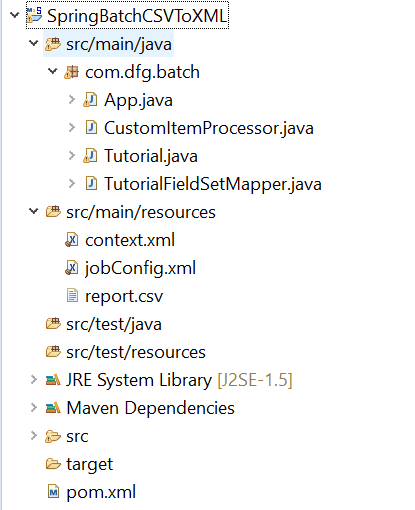
**<tutorial\_title>Learn JavaFX</tutorial\_title>**

**</details>**

**</Tutorial>**

# **Spring Batch - CSV to XML**

* **Project Structure :**



* **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.dfg.batch</groupId>

<artifactId> SpringBatchCSVToXML</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name> SpringBatchCSVToXML</name>

<properties>

<jdk.version>1.6</jdk.version>

<spring.version>4.0.0.RELEASE</spring.version>

<spring.batch.version>3.0.7.RELEASE</spring.batch.version>

<mysql.driver.version>5.1.25</mysql.driver.version>

<junit.version>4.11</junit.version>

</properties>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring jdbc, for database -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring XML to/back object -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- MySQL database driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.driver.version}</version>

</dependency>

<!-- Spring Batch dependencies -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-core</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-infrastructure</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<!-- Spring Batch unit test -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-test</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.apache.pdfbox</groupId>

<artifactId>pdfbox</artifactId>

<version>2.0.8</version>

</dependency>

<!-- Junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<finalName> SpringBatchCSVToXML</finalName>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-eclipse-plugin</artifactId>

<version>2.9</version>

<configuration>

<downloadSources>true</downloadSources>

<downloadJavadocs>false</downloadJavadocs>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Reader** − The **reader** we are using in the application is **FlatFileItemReader** to read data from the CSV files.

Following is the input CSV file we are using in this application. This document holds data records which specify details like tutorial id, tutorial author, tutorial title, submission date, tutorial icon and tutorial description.

1001, "Sanjay", "Learn Java", 06/05/2007

1002, "Abdul S", "Learn MySQL", 19/04/2007

1003, "Krishna Kasyap", "Learn JavaFX", 06/07/2017

**Writer** − The Writer we are using in the application is **StaxEventItemWriter** to write the data to XML file.

**Processor** − The Processor we are using in the application is a custom processor which just prints the records read from the CSV file.

* **jobConfig.xml:**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:batch = *"http://www.springframework.org/schema/batch"*

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

xmlns:util = *"http://www.springframework.org/schema/util"*

xsi:schemaLocation = *"http://www.springframework.org/schema/batch*

*http://www.springframework.org/schema/batch/spring-batch-2.2.xsd*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/util*

*http://www.springframework.org/schema/util/spring-util-3.0.xsd "*>

<import resource = *"context.xml"* />

<!-- Create Spring Batch Job -->

<batch:job id = *"csvtoxmljob"*>

<batch:step id = *"step1"*>

<batch:tasklet>

<batch:chunk reader = *"cvsFileItemReader"*

writer = *"xmlItemWriter"* processor = *"itemProcessor"* commit-interval = *"10"*>

</batch:chunk>

</batch:tasklet>

</batch:step>

</batch:job>

<bean id = *"cvsFileItemReader"*

class = *"org.springframework.batch.item.file.FlatFileItemReader"*>

<property name = *"resource"* value = *"classpath:report.csv"* />

<property name = *"lineMapper"*>

<bean

class = *"org.springframework.batch.item.file.mapping.DefaultLineMapper"*>

<property name = *"lineTokenizer"*>

<bean

class = *"org.springframework.batch.item.file.transform.DelimitedLineTokenizer"*>

<property name = *"names"* value = *"tutorial\_id,*

*tutorial\_author, Tutorial\_title, submission\_date"* />

</bean>

</property>

<property name = *"fieldSetMapper"*>

<bean class = *"com.dfg.batch.TutorialFieldSetMapper"* />

</property>

</bean>

</property>

</bean>

<bean id = *"xmlItemWriter"*

class = *"org.springframework.batch.item.xml.StaxEventItemWriter"*>

<property name = *"resource"* value = *"file:tutorials.xml"* />

<property name = *"marshaller"* ref = *"reportMarshaller"* />

<property name = *"rootTagName"* value = *"tutorials"* />

</bean>

<bean id = *"reportMarshaller"*

class = *"org.springframework.oxm.jaxb.Jaxb2Marshaller"*>

<property name = *"classesToBeBound"*>

<list>

<value>com.dfg.batch.Tutorial</value>

</list>

</property>

</bean>

<bean id = *"report"* class = *"Report"* scope = *"prototype"* />

<bean id = *"itemProcessor"* class = *"com.dfg.batch.CustomItemProcessor"* />

</beans>

* **context.xml:**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:jdbc = *"http://www.springframework.org/schema/jdbc"*

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

xsi:schemaLocation = *"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/jdbc*

*http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd"*>

<!-- Create JobRepository -->

<bean id = *"jobRepository"*

class = *"org.springframework.batch.core.repository.support.JobRepositoryFactoryBean"*>

<property name = *"dataSource"* ref = *"dataSource"* /> <!-- Inject DataSource -->

<property name = *"transactionManager"* ref = *"transactionManager"* /> <!-- Inject Txn Manager -->

<property name = *"databaseType"* value = *"mysql"* />

</bean>

<!-- Create Txn Manager -->

<bean id = *"transactionManager"*

class = *"org.springframework.batch.support.transaction.ResourcelessTransactionManager"* />

<!-- Create DataSource -->

<bean id = *"dataSource"*

class = *"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name = *"driverClassName"* value = *"com.mysql.jdbc.Driver"* />

<property name = *"url"* value = *"jdbc:mysql://localhost:3306/springorm"* />

<property name = *"username"* value = *"root"* />

<property name = *"password"* value = *"123456"* />

</bean>

<!-- Create Job Launcher -->

<bean id = *"jobLauncher"*

class = *"org.springframework.batch.core.launch.support.SimpleJobLauncher"*>

<property name = *"jobRepository"* ref = *"jobRepository"* />

</bean>

<!-- create job-meta tables automatically -->

<jdbc:initialize-database data-source = *"dataSource"*>

<jdbc:script location = *"org/springframework/batch/core/schema-drop-mysql.sql"*/>

<jdbc:script location = *"org/springframework/batch/core/schema-mysql.sql"*/>

</jdbc:initialize-database>

</beans>

* Tutorial.java:

**package** com.dfg.batch;

**import** javax.xml.bind.annotation.XmlAttribute;

**import** javax.xml.bind.annotation.XmlElement;

**import** javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "tutorials")

**public** **class** Tutorial {

**int** tutorial\_id;

String tutorial\_author;

String submission\_date;

String tutorial\_title;

@XmlAttribute(name = "tutorial\_id")

**public** **int** getTutorial\_id() {

**return** tutorial\_id;

}

**public** **void** setTutorial\_id(**int** tutorial\_id) {

**this**.tutorial\_id = tutorial\_id;

}

@XmlElement(name = "tutorial\_author")

**public** String getTutorial\_author() {

**return** tutorial\_author;

}

**public** **void** setTutorial\_author(String tutorial\_author) {

**this**.tutorial\_author = tutorial\_author;

}

@XmlElement(name = "tutorial\_title")

**public** String getTutorial\_title() {

**return** tutorial\_title;

}

**public** **void** setTutorial\_title(String tutorial\_title) {

**this**.tutorial\_title = tutorial\_title;

}

@XmlElement(name = "submission\_date")

**public** String getSubmission\_date() {

**return** submission\_date;

}

**public** **void** setSubmission\_date(String submission\_date) {

**this**.submission\_date = submission\_date;

}

**public** String toString() {

**return** " [Tutorial Id=" + tutorial\_id + ",Tutorial Author =" + tutorial\_author + ",Tutorial Title =" + tutorial\_title + ",Submission Date =" + submission\_date + "]";

}

}

* CustomItemProcessor.java

**package** com.dfg.batch;

**import** org.springframework.batch.item.ItemProcessor;

//This class acts as Item Processor

**public** **class** CustomItemProcessor **implements** ItemProcessor<Tutorial, Tutorial> {

**public** Tutorial process(Tutorial item) **throws** Exception {

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

**return** item;

}

}

* **TutorialFieldSetMapper.java**

**package** com.dfg.batch;

**import** org.springframework.batch.item.file.mapping.FieldSetMapper;

**import** org.springframework.batch.item.file.transform.FieldSet;

**import** org.springframework.validation.BindException;

**public** **class** TutorialFieldSetMapper **implements** FieldSetMapper<Tutorial> {

**public** Tutorial mapFieldSet(FieldSet fieldSet) **throws** BindException {

//Instantiating the report object

Tutorial tutorial = **new** Tutorial();

//Setting the fields

tutorial.setTutorial\_id(fieldSet.readInt(0));

tutorial.setTutorial\_author(fieldSet.readString(1));

tutorial.setTutorial\_title(fieldSet.readString(2));

tutorial.setSubmission\_date(fieldSet.readString(3));

**return** tutorial;

}

}

* **App.java:**

**package** com.dfg.batch;

**import** org.springframework.batch.core.Job;

**import** org.springframework.batch.core.JobExecution;

**import** org.springframework.batch.core.JobParameters;

**import** org.springframework.batch.core.launch.JobLauncher;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** App {

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

String springConfig = "jobConfig.xml";

// Creating the application context object

ApplicationContext context = **new** ClassPathXmlApplicationContext(springConfig);

// Creating the job launcher

JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

// Creating the job

Job job = (Job) context.getBean("csvtoxmljob");

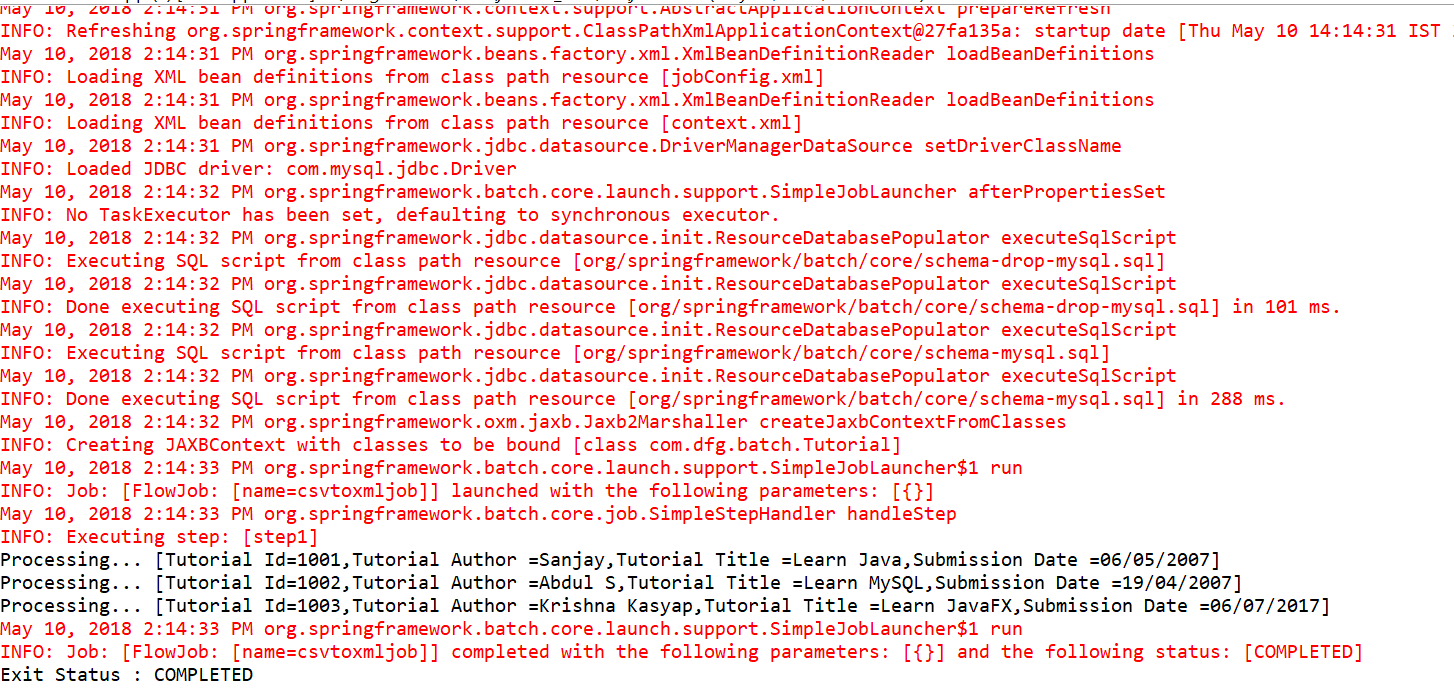
// Executing the JOB

JobExecution execution = jobLauncher.run(job, **new** JobParameters());

System.***out***.println("Exit Status : " + execution.getStatus());

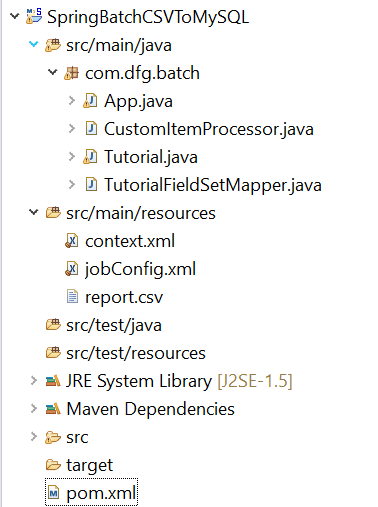
}

}



# **Spring Batch - CSV to MySQL**

* **Project Structure :**



* **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.dfg.batch</groupId>

<artifactId>SpringBatchXMLToMySQL</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBatchXMLToMySQL</name>

<properties>

<jdk.version>1.6</jdk.version>

<spring.version>4.0.0.RELEASE</spring.version>

<spring.batch.version>3.0.7.RELEASE</spring.batch.version>

<mysql.driver.version>5.1.25</mysql.driver.version>

<junit.version>4.11</junit.version>

</properties>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring jdbc, for database -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring XML to/back object -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- MySQL database driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.driver.version}</version>

</dependency>

<!-- Spring Batch dependencies -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-core</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-infrastructure</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<!-- Spring Batch unit test -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-test</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.apache.pdfbox</groupId>

<artifactId>pdfbox</artifactId>

<version>2.0.8</version>

</dependency>

<!-- Junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<finalName>SpringBatchXMLToMySQL</finalName>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-eclipse-plugin</artifactId>

<version>2.9</version>

<configuration>

<downloadSources>true</downloadSources>

<downloadJavadocs>false</downloadJavadocs>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Reader** − The **reader** we are using in the application is **FlatFileItemReader** to read data from the CSV files.

Following is the input CSV file we are using in this application. This document holds data records which specify details like tutorial id, tutorial author, tutorial title, submission date, tutorial icon and tutorial description.

1004, "Kasturi Bhalerao", "Learn Spring", 10/05/2018

**Writer** −The **writer** we are using in the application is **JdbcBatchItemWriter** to write the data to MySQL database. Assume we have created a table in MySQL inside a table called **"** TUTORIALS **"**.

CREATE TABLE TUTORIALS(

tutorial\_id int(10) NOT NULL,

tutorial\_author VARCHAR(20),

tutorial\_title VARCHAR(50),

submission\_date VARCHAR(20),

);

* **jobConfig.xml:**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:batch = *"http://www.springframework.org/schema/batch"*

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

xmlns:util = *"http://www.springframework.org/schema/util"*

xsi:schemaLocation = *"http://www.springframework.org/schema/batch*

*http://www.springframework.org/schema/batch/spring-batch-2.2.xsd*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/util*

*http://www.springframework.org/schema/util/spring-util-3.0.xsd "*>

<import resource = *"context.xml"* />

<!-- Create Spring Batch Job -->

<batch:job id = *"csvtomysqljob"*>

<batch:step id = *"step1"*>

<batch:tasklet>

<batch:chunk reader = *"cvsFileItemReader"*

writer = *"mysqlItemWriter"* processor = *"itemProcessor"* commit-interval = *"10"*>

</batch:chunk>

</batch:tasklet>

</batch:step>

</batch:job>

<bean id = *"cvsFileItemReader"*

class = *"org.springframework.batch.item.file.FlatFileItemReader"*>

<property name = *"resource"* value = *"classpath:report.csv"* />

<property name = *"lineMapper"*>

<bean

class = *"org.springframework.batch.item.file.mapping.DefaultLineMapper"*>

<property name = *"lineTokenizer"*>

<bean

class = *"org.springframework.batch.item.file.transform.DelimitedLineTokenizer"*>

<property name = *"names"* value = *"tutorial\_id,*

*tutorial\_author, Tutorial\_title, submission\_date"* />

</bean>

</property>

<property name = *"fieldSetMapper"*>

<bean class = *"com.dfg.batch.TutorialFieldSetMapper"* />

</property>

</bean>

</property>

</bean>

<bean id = *"mysqlItemWriter"* class = *"org.springframework.batch.item.database.JdbcBatchItemWriter"*>

<property name = *"dataSource"* ref = *"dataSource"* />

<property name = *"sql"*>

<value>

<![CDATA[insert into tutorials (tutorial\_id, tutorial\_author, tutorial\_title,

submission\_date)

values (:tutorial\_id, :tutorial\_author, :tutorial\_title, :submission\_date);]]>

</value>

</property>

<property name = *"itemSqlParameterSourceProvider"*>

<bean class = *"org.springframework.batch.item.database.BeanPropertyItemSqlParameterSourceProvider"* />

</property>

</bean>

<bean id = *"reportMarshaller"*

class = *"org.springframework.oxm.jaxb.Jaxb2Marshaller"*>

<property name = *"classesToBeBound"*>

<list>

<value>com.dfg.batch.Tutorial</value>

</list>

</property>

</bean>

<bean id = *"report"* class = *"Report"* scope = *"prototype"* />

<bean id = *"itemProcessor"* class = *"com.dfg.batch.CustomItemProcessor"* />

</beans>

* context.xml:

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:jdbc = *"http://www.springframework.org/schema/jdbc"*

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

xsi:schemaLocation = *"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/jdbc*

*http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd"*>

<!-- Create JobRepository -->

<bean id = *"jobRepository"*

class = *"org.springframework.batch.core.repository.support.JobRepositoryFactoryBean"*>

<property name = *"dataSource"* ref = *"dataSource"* /> <!-- Inject DataSource -->

<property name = *"transactionManager"* ref = *"transactionManager"* /> <!-- Inject Txn Manager -->

<property name = *"databaseType"* value = *"mysql"* />

</bean>

<!-- Create Txn Manager -->

<bean id = *"transactionManager"*

class = *"org.springframework.batch.support.transaction.ResourcelessTransactionManager"* />

<!-- Create DataSource -->

<bean id = *"dataSource"*

class = *"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name = *"driverClassName"* value = *"com.mysql.jdbc.Driver"* />

<property name = *"url"* value = *"jdbc:mysql://localhost:3306/springorm"* />

<property name = *"username"* value = *"root"* />

<property name = *"password"* value = *"123456"* />

</bean>

<!-- Create Job Launcher -->

<bean id = *"jobLauncher"*

class = *"org.springframework.batch.core.launch.support.SimpleJobLauncher"*>

<property name = *"jobRepository"* ref = *"jobRepository"* />

</bean>

<!-- create job-meta tables automatically -->

<jdbc:initialize-database data-source = *"dataSource"*>

<jdbc:script location = *"org/springframework/batch/core/schema-drop-mysql.sql"*/>

<jdbc:script location = *"org/springframework/batch/core/schema-mysql.sql"*/>

</jdbc:initialize-database>

</beans>

* **Tutorial.java:**

**package** com.dfg.batch;

**import** javax.xml.bind.annotation.XmlAttribute;

**import** javax.xml.bind.annotation.XmlElement;

**import** javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "tutorials")

**public** **class** Tutorial {

**int** tutorial\_id;

String tutorial\_author;

String submission\_date;

String tutorial\_title;

@XmlAttribute(name = "tutorial\_id")

**public** **int** getTutorial\_id() {

**return** tutorial\_id;

}

**public** **void** setTutorial\_id(**int** tutorial\_id) {

**this**.tutorial\_id = tutorial\_id;

}

@XmlElement(name = "tutorial\_author")

**public** String getTutorial\_author() {

**return** tutorial\_author;

}

**public** **void** setTutorial\_author(String tutorial\_author) {

**this**.tutorial\_author = tutorial\_author;

}

@XmlElement(name = "tutorial\_title")

**public** String getTutorial\_title() {

**return** tutorial\_title;

}

**public** **void** setTutorial\_title(String tutorial\_title) {

**this**.tutorial\_title = tutorial\_title;

}

@XmlElement(name = "submission\_date")

**public** String getSubmission\_date() {

**return** submission\_date;

}

**public** **void** setSubmission\_date(String submission\_date) {

**this**.submission\_date = submission\_date;

}

**public** String toString() {

**return** " [Tutorial Id=" + tutorial\_id + ",Tutorial Author =" + tutorial\_author + ",Tutorial Title =" + tutorial\_title + ",Submission Date =" + submission\_date + "]";

}

}

* **TutorialFieldSetMapper.java:**

**package** com.dfg.batch;

**import** org.springframework.batch.item.file.mapping.FieldSetMapper;

**import** org.springframework.batch.item.file.transform.FieldSet;

**import** org.springframework.validation.BindException;

**public** **class** TutorialFieldSetMapper **implements** FieldSetMapper<Tutorial> {

**public** Tutorial mapFieldSet(FieldSet fieldSet) **throws** BindException {

//Instantiating the report object

Tutorial tutorial = **new** Tutorial();

//Setting the fields

tutorial.setTutorial\_id(fieldSet.readInt(0));

tutorial.setTutorial\_author(fieldSet.readString(1));

tutorial.setTutorial\_title(fieldSet.readString(2));

tutorial.setSubmission\_date(fieldSet.readString(3));

**return** tutorial;

}

}

* **CustomItemProcessor.java:**

**package** com.dfg.batch;

**import** org.springframework.batch.item.ItemProcessor;

//This class acts as Item Processor

**public** **class** CustomItemProcessor **implements** ItemProcessor<Tutorial, Tutorial> {

**public** Tutorial process(Tutorial item) **throws** Exception {

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

**return** item;

}

}

* **App.java:**

**package** com.dfg.batch;

**import** org.springframework.batch.core.Job;

**import** org.springframework.batch.core.JobExecution;

**import** org.springframework.batch.core.JobParameters;

**import** org.springframework.batch.core.launch.JobLauncher;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** App {

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

String springConfig = "jobConfig.xml";

// Creating the application context object

ApplicationContext context = **new** ClassPathXmlApplicationContext(springConfig);

// Creating the job launcher

JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

// Creating the job

Job job = (Job) context.getBean("csvtomysqljob");

// Executing the JOB

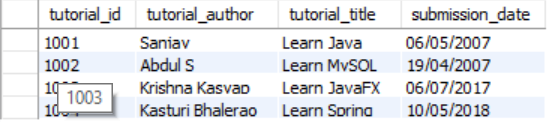
JobExecution execution = jobLauncher.run(job, **new** JobParameters());

System.***out***.println("Exit Status : " + execution.getStatus());

}

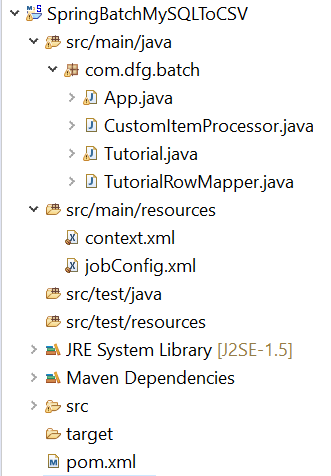
}

Run the App.java using Java Application



# **Spring Batch - MySQL to CSV**

* **Project Structure :**



* **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.dfg.batch</groupId>

<artifactId>SpringBatchMySQLToCSV</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>SpringBatchMySQLToCSV</name>

<properties>

<jdk.version>1.6</jdk.version>

<spring.version>4.0.0.RELEASE</spring.version>

<spring.batch.version>3.0.7.RELEASE</spring.batch.version>

<mysql.driver.version>5.1.25</mysql.driver.version>

<junit.version>4.11</junit.version>

</properties>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring jdbc, for database -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-jdbc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring XML to/back object -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-oxm</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- MySQL database driver -->

<dependency>

<groupId>mysql</groupId>

<artifactId>mysql-connector-java</artifactId>

<version>${mysql.driver.version}</version>

</dependency>

<!-- Spring Batch dependencies -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-core</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-infrastructure</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<!-- Spring Batch unit test -->

<dependency>

<groupId>org.springframework.batch</groupId>

<artifactId>spring-batch-test</artifactId>

<version>${spring.batch.version}</version>

</dependency>

<dependency>

<groupId>org.apache.pdfbox</groupId>

<artifactId>pdfbox</artifactId>

<version>2.0.8</version>

</dependency>

<!-- Junit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>${junit.version}</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<finalName>SpringBatchMySQLToCSV</finalName>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-eclipse-plugin</artifactId>

<version>2.9</version>

<configuration>

<downloadSources>true</downloadSources>

<downloadJavadocs>false</downloadJavadocs>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Reader** − The reader we are using in the application is **JdbcCursorItemReader** to read data from MySQL database.

Assume we have created a table in the MySQL database as shown below −

CREATE TABLE `tutorials` (

`tutorial\_id` int(10) NOT NULL,

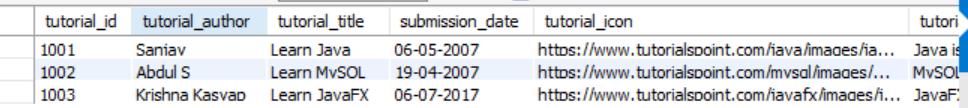
`tutorial\_author` varchar(20) DEFAULT NULL,

`tutorial\_title` varchar(50) DEFAULT NULL,

`submission\_date` varchar(20) DEFAULT NULL,

) ;

Assume we have inserted the following records in to it.



**Writer** − The Writer we are using in the application is **FlatFileItemWriter** to write the data to **flatfile** (.txt).

**Processor** − The Processor we are using in the application is a custom processor which just prints the records read from the CSV file.

* **jobConfig.xml:**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:batch = *"http://www.springframework.org/schema/batch"*

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

xmlns:util = *"http://www.springframework.org/schema/util"*

xsi:schemaLocation = *"http://www.springframework.org/schema/batch*

*http://www.springframework.org/schema/batch/spring-batch-2.2.xsd*

*http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/util*

*http://www.springframework.org/schema/util/spring-util-3.0.xsd "*>

<import resource = *"context.xml"* />

<!-- Create Spring Batch Job -->

<batch:job id = *"mysqltocsvjob"*>

<batch:step id = *"step1"*>

<batch:tasklet>

<batch:chunk reader = *"mysqlItemReader"*

writer = *"flatFileItemWriter"* processor = *"itemProcessor"* commit-interval = *"10"*>

</batch:chunk>

</batch:tasklet>

</batch:step>

</batch:job>

<bean id = *"mysqlItemReader"*

class = *"org.springframework.batch.item.database.JdbcCursorItemReader"* >

<property name = *"dataSource"* ref = *"dataSource"* />

<property name = *"sql"* value = *"select \* from tutorials"* />

<property name = *"rowMapper"*>

<bean class = *"com.dfg.batch.TutorialRowMapper"* />

</property>

</bean>

<bean id = *"flatFileItemWriter"*

class = *" org.springframework.batch.item.file.FlatFileItemWriter"*>

<property name = *"resource"* value = *"file:tutorials.txt"*/>

<property name = *"lineAggregator"*>

<bean class = *" org.springframework.batch.item.file.transform.PassThroughLineAggregator"*/>

</property>

</bean>

<bean id = *"itemProcessor"* class = *"com.dfg.batch.CustomItemProcessor"* />

</beans>

* **context.xml:**

<beans xmlns = *"http://www.springframework.org/schema/beans"*

xmlns:jdbc = *"http://www.springframework.org/schema/jdbc"*

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

xsi:schemaLocation = *"http://www.springframework.org/schema/beans*

*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*

*http://www.springframework.org/schema/jdbc*

*http://www.springframework.org/schema/jdbc/spring-jdbc-3.2.xsd"*>

<!-- Create JobRepository -->

<bean id = *"jobRepository"*

class = *"org.springframework.batch.core.repository.support.JobRepositoryFactoryBean"*>

<property name = *"dataSource"* ref = *"dataSource"* /> <!-- Inject DataSource -->

<property name = *"transactionManager"* ref = *"transactionManager"* /> <!-- Inject Txn Manager -->

<property name = *"databaseType"* value = *"mysql"* />

</bean>

<!-- Create Txn Manager -->

<bean id = *"transactionManager"*

class = *"org.springframework.batch.support.transaction.ResourcelessTransactionManager"* />

<!-- Create DataSource -->

<bean id = *"dataSource"*

class = *"org.springframework.jdbc.datasource.DriverManagerDataSource"*>

<property name = *"driverClassName"* value = *"com.mysql.jdbc.Driver"* />

<property name = *"url"* value = *"jdbc:mysql://localhost:3306/springorm"* />

<property name = *"username"* value = *"root"* />

<property name = *"password"* value = *"123456"* />

</bean>

<!-- Create Job Launcher -->

<bean id = *"jobLauncher"*

class = *"org.springframework.batch.core.launch.support.SimpleJobLauncher"*>

<property name = *"jobRepository"* ref = *"jobRepository"* />

</bean>

<!-- create job-meta tables automatically -->

<jdbc:initialize-database data-source = *"dataSource"*>

<jdbc:script location = *"org/springframework/batch/core/schema-drop-mysql.sql"*/>

<jdbc:script location = *"org/springframework/batch/core/schema-mysql.sql"*/>

</jdbc:initialize-database>

</beans>

* **Tutorial.java:**

**package** com.dfg.batch;

**import** javax.xml.bind.annotation.XmlAttribute;

**import** javax.xml.bind.annotation.XmlElement;

**import** javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement(name = "details")

**public** **class** Tutorial {

**int** tutorial\_id;

String tutorial\_author;

String submission\_date;

String tutorial\_title;

@XmlAttribute(name = "tutorial\_id")

**public** **int** getTutorial\_id() {

**return** tutorial\_id;

}

**public** **void** setTutorial\_id(**int** tutorial\_id) {

**this**.tutorial\_id = tutorial\_id;

}

@XmlElement(name = "tutorial\_author")

**public** String getTutorial\_author() {

**return** tutorial\_author;

}

**public** **void** setTutorial\_author(String tutorial\_author) {

**this**.tutorial\_author = tutorial\_author;

}

@XmlElement(name = "tutorial\_title")

**public** String getTutorial\_title() {

**return** tutorial\_title;

}

**public** **void** setTutorial\_title(String tutorial\_title) {

**this**.tutorial\_title = tutorial\_title;

}

@XmlElement(name = "submission\_date")

**public** String getSubmission\_date() {

**return** submission\_date;

}

**public** **void** setSubmission\_date(String submission\_date) {

**this**.submission\_date = submission\_date;

}

**public** String toString() {

**return** " [Tutorial Id=" + tutorial\_id + ",Tutorial Author =" + tutorial\_author + ",Tutorial Title =" + tutorial\_title + ",Submission Date =" + submission\_date + "]";

}

}

* **TutorialRowMapper.java:**

**package** com.dfg.batch;

**import** java.sql.ResultSet;

**import** java.sql.SQLException;

**import** org.springframework.jdbc.core.RowMapper;

**public** **class** TutorialRowMapper **implements** RowMapper<Tutorial> {

**public** Tutorial mapRow(ResultSet rs, **int** rowNum) **throws** SQLException {

Tutorial tutorial = **new** Tutorial();

tutorial.setTutorial\_id(rs.getInt("tutorial\_id"));

tutorial.setTutorial\_author(rs.getString("tutorial\_author"));

tutorial.setTutorial\_title(rs.getString("tutorial\_title"));

tutorial.setSubmission\_date(rs.getString("submission\_date"));

**return** tutorial;

}

}

* **CustomItemProcessor.java**

**package** com.dfg.batch;

**import** org.springframework.batch.item.ItemProcessor;

//This class acts as Item Processor

**public** **class** CustomItemProcessor **implements** ItemProcessor<Tutorial, Tutorial> {

**public** Tutorial process(Tutorial item) **throws** Exception {

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

**return** item;

}

}

* **App.java**

**package** com.dfg.batch;

**import** org.springframework.batch.core.Job;

**import** org.springframework.batch.core.JobExecution;

**import** org.springframework.batch.core.JobParameters;

**import** org.springframework.batch.core.launch.JobLauncher;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** App {

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

String springConfig = "jobConfig.xml";

// Creating the application context object

ApplicationContext context = **new** ClassPathXmlApplicationContext(springConfig);

// Creating the job launcher

JobLauncher jobLauncher = (JobLauncher) context.getBean("jobLauncher");

// Creating the job

Job job = (Job) context.getBean("mysqltocsvjob");

// Executing the JOB

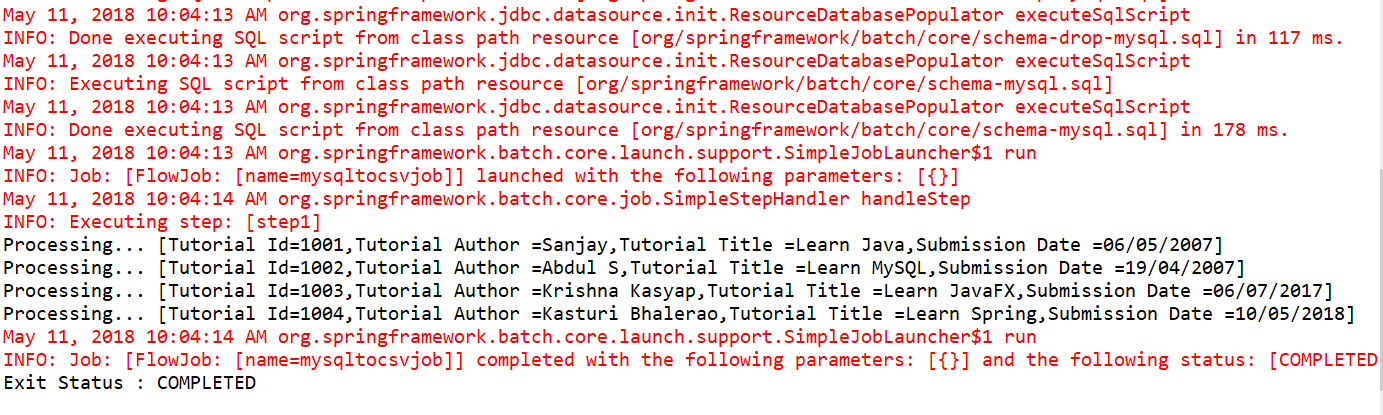
JobExecution execution = jobLauncher.run(job, **new** JobParameters());

System.***out***.println("Exit Status : " + execution.getStatus());

}

}

**Console:**



**Tutorial.txt :**

