

Features implemented by Spring Batch include data validation, formatting of output, the ability to implement complex business rules in a reusable way, and the ability to handle large data sets The application layer spends most of its time interacting with the next layer, the core. The core layer contains all the pieces that define the batch domain. Elements of the core component include the Job and Step interfaces as well as the interfaces used to execute a Job: JobLauncher and JobParameters.

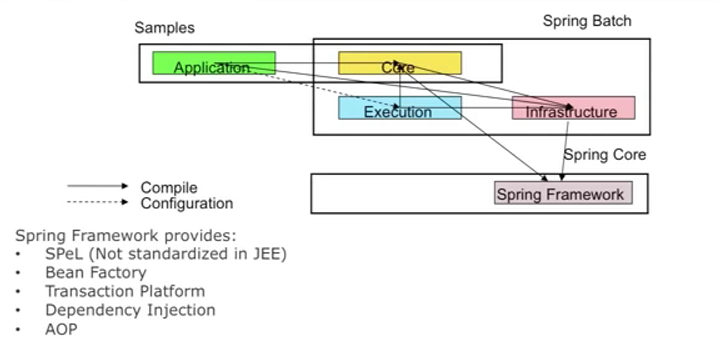
Below all this is the infrastructure layer . In order to do any processing, you need to read and write from files, databases, and so on. You must be able to handle what to do when a job is retried after a failure. These pieces are considered common infrastructure and live in the infrastructure component of the framework.

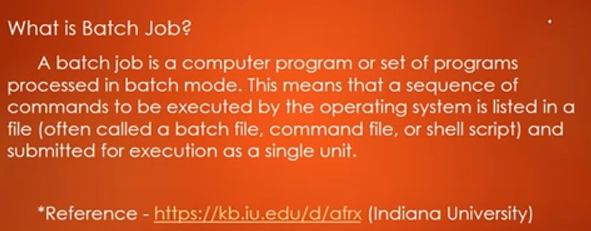
A common misconception is that Spring Batch is or has a scheduler. It doesn't. There is no way within the framework to schedule a job to run at a given time or based on a given event. There are a number of ways to launch a job, from a simple cron script to Quartz or even an enterprise scheduler like UC4, but none within the framework itself.

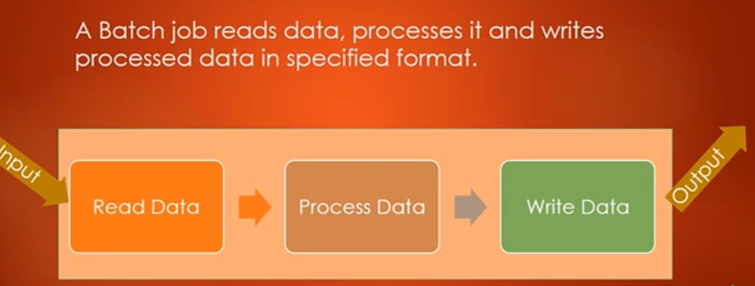
It's one thing to be able to write a Java program that processes some data once and never runs again. But mission-critical processes require a more robust approach. The ability to keep the state of a job for re-execution, maintaining data integrity when a job fails through transaction management and saving performance metrics of past job executions for trending, are features that you expect in an enterprise batch system. These features are included in Spring Batch, and most of them are turned on by default; they require only minimal tweaking for performance and requirements as you develop your process.

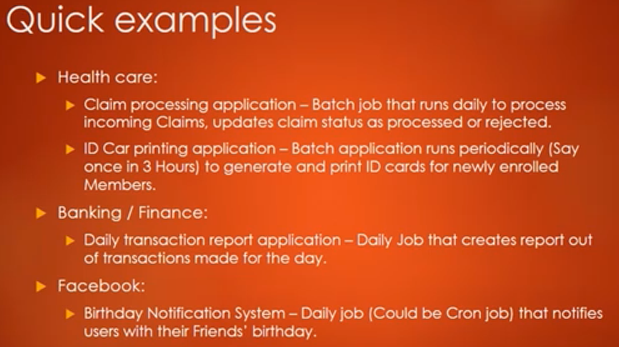
Parallel chunk/step processing, remote chunk processing, and partitioning

Spring batch implementation of jsr352



s



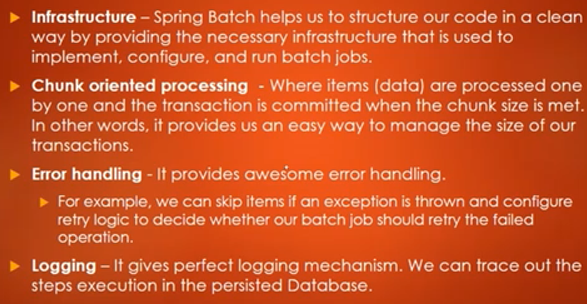


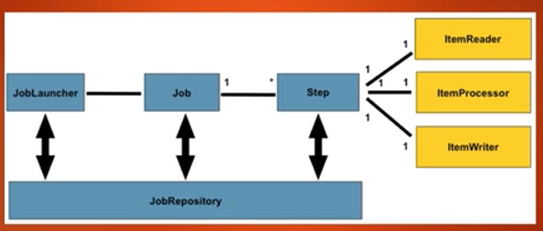
Job that runs every day/ periodically

Scheduling of job

**Spring batch Architecture**

****

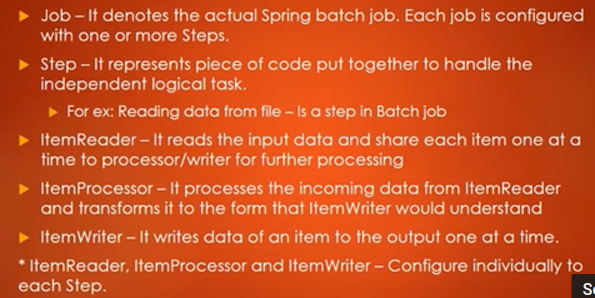
****

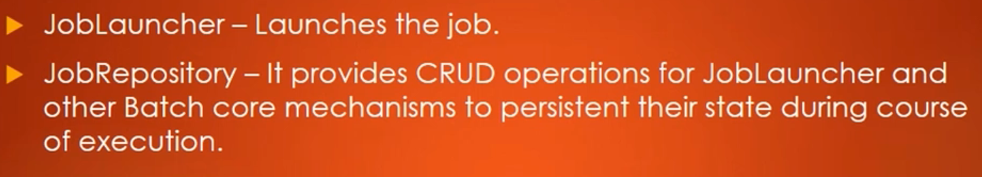
****

**Job Launcher: Triggers job picking info from repo**

JobRepository: job info in db.

Any number of steps in a job: send email or processing data (read data, process it and write it)



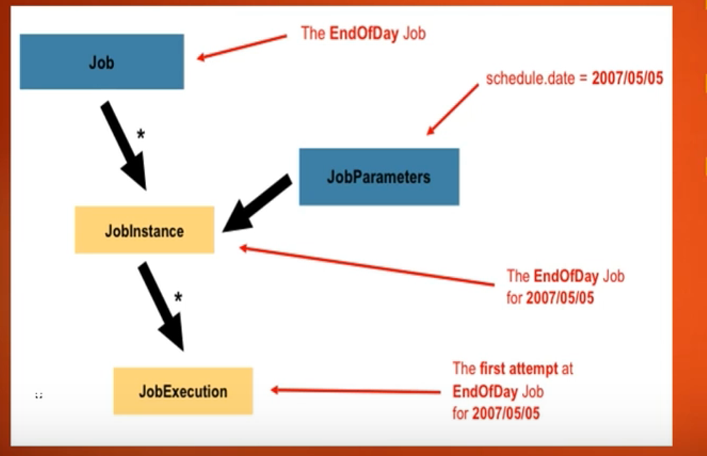


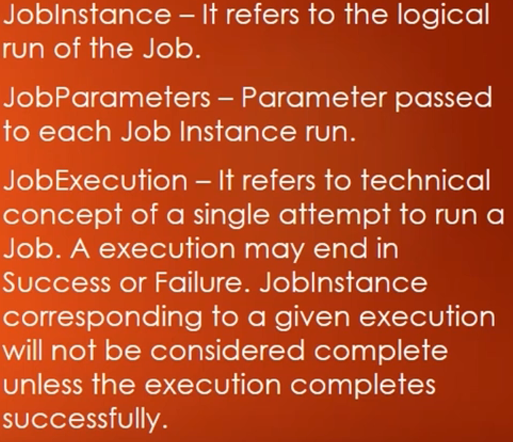
Step1: read data and write to db

Step 2: Send an email

Step 3: Send a report from db

While step 1 is executing, updates job repository the status of the job. On completion, will set it to step 1 is success. And step 2 as running.





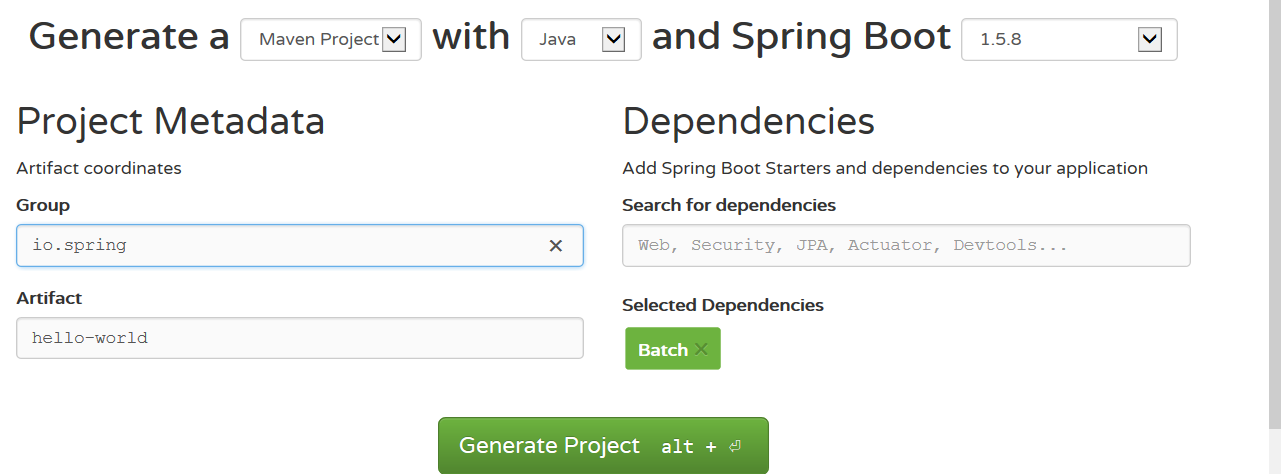
Spring batch : is called job. Has many number of steps. Each step takes care of specific individual task.

Batch processing: processing of finite amount of data without interaction or interruption. Whether it be a web based app or a messaging app, there is a code that sits on top of the server listening , waiting for some type of external stimulus like in a web app, can be an incoming request, in messaging app could be a message but in batch processing , there is nothing like an external stimulus.

1. Job flow state machine: Transitioning from 1 step to other and configuring them in xml or java based dsl
2. Transaction handling: Not all data can be processed together. So putting them into chunks and maintaining states so that as we commit incrementally, we can restart at the point we failed
3. Declarative IO: Reading from a file/ jdbc, xml and writing to it. So that you can focus on business logic rather than how to read/ write
4. Robust error handling:
5. Scalability: To scale from 1 jvm to multiple jvms
6. Battle tested: handle any size workloads
7. Built on spring: can use bootstrapping, spring boot, context, spring initialize, all of IDE integrations, testing capabilities

**Project set-up**

**Start.spring.io**

****

Download zip. Extract. Import existing maven project in Eclipse ide.

Application.properties file: property file for spring boot application

**A step has a reader, writer and a processor or can be just a tasklet where it does not have a reader/ writer**

A *tasklet* is a special

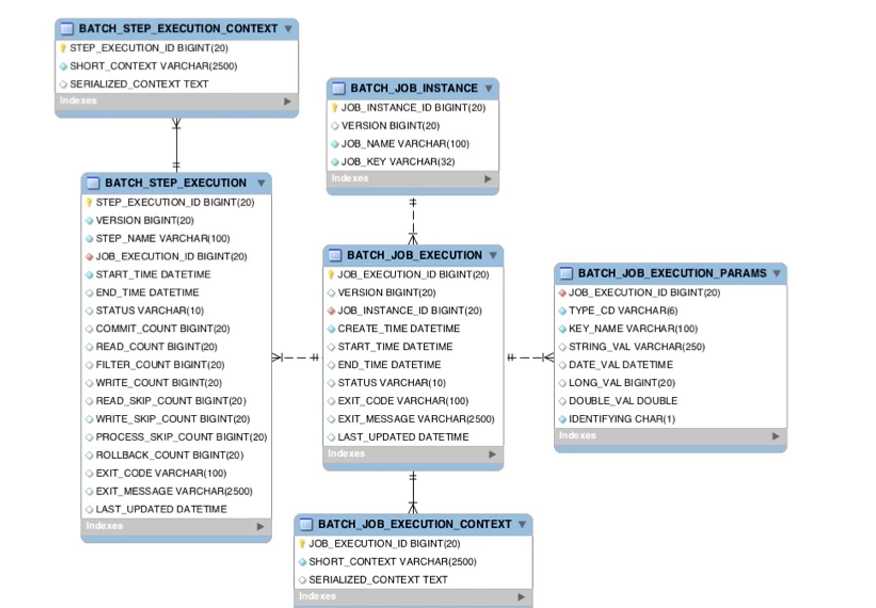
type of step that is used to perform a function without a reader or writer. Typically, a tasklet is used for a

single function, say performing some initialization, calling a stored procedure, or sending an e-mail to

alert you that the job has finished.

Spring provides 2 implementations of job repositories to store states, steps of job

1. Map based : uses hashmap (in-memory job repository, default): Not thread safe, not used for production env
2. JDBC based



**Batch status**

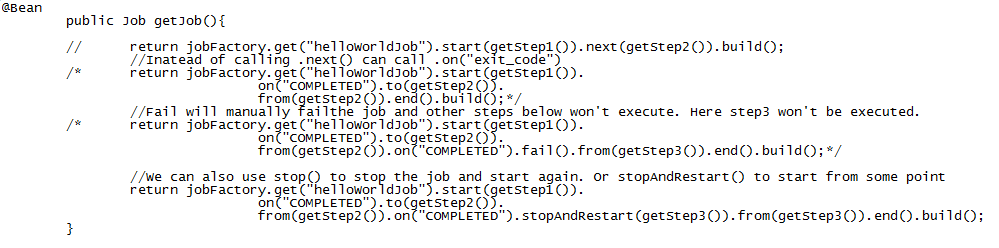
****

**Exit\_code**

We can programmatically set the value for it. Else by-default it is set as the status of the job.

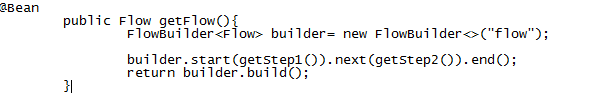
**Exit\_message**

Also used for debugging purpose. If an exception is thrown, this message will be populated by the stack trace.



**Flows:** Sequence of steps , components can be reused. Flows can be reused in steps. Like initialization steps can be configured in a flow and added to many jobs.

**Configuring a flow:**

****

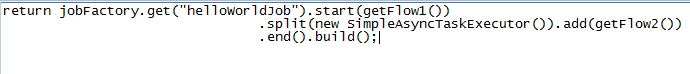
**Can use flow now in a job**

****

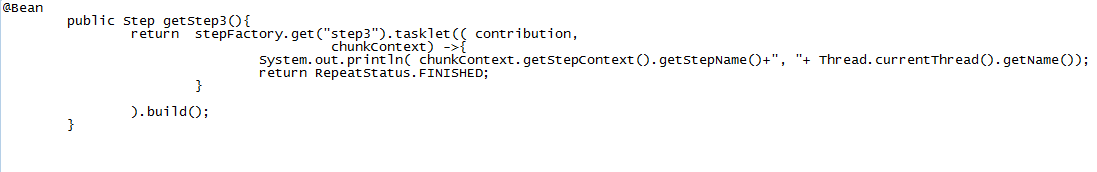
**By default all jobs written would be executed by spring**

**Split:**

Split a job in two or more independent parallel flows



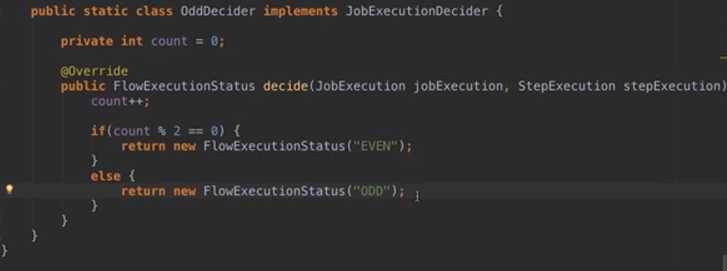
Can modify steps to print thread name and step name

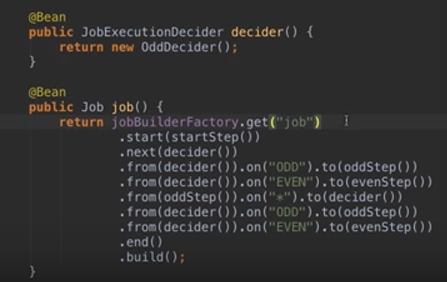


Split can be used to import the data in two different tables together.

**Decision Job Flow**

To take decisions, to write complex logic to decide as to what the next step should be.





If you return RepeatStatus.CONTINUABLE , you aren't saying that the job can continue. You're

telling Spring Batch to run the tasklet again. Say, for example, that you wanted to execute a particular

tasklet in a loop until a given condition was met, yet you still wanted to use Spring Batch to keep track of

how many times the tasklet was executed, transactions, and so on. Your tasklet could return

RepeatStatus.CONTINUABLE until the condition was met. If you return RepeatStatus.FINISHED , that means

the processing for this tasklet is complete (regardless of success) and to continue with the next piece of

processing.

**Running Spring Job**

Spring Batch comes with its own job runner called CommandLineJobRunner. As you can guess, it's intended to be run from … a command line! In this topic, you will execute your jobs from your project's target directory so that you won't need to go through setting up the classpath. The CommandLineJobRunner takes two or more parameters: the path to the XML file that contains the job configuration, the name of the job to be executed, and a list of job parameters.

java -jar hello-world-0.0.1-SNAPSHOT.jar jobs/helloWorld.xml helloWorldJob

**JobRepository Configuration:**

To change where Spring Batch stores the data, you need to do three things: update the batch.properties file, update your pom, and create the batch schema in your database. 3 Let's start by modifying the batch.properties file found in your project's /src/main/resources directory. The properties should be pretty straightforward. Listing 2-7 shows what I have in mine.

**batch.properties File**

batch.jdbc.driver=com.mysql.jdbc.Driver

batch.jdbc.url=jdbc:mysql://localhost:3306/spring\_batch\_test

# use this one for a separate server process so you can inspect the results

# (or add it to system properties with -D to override at run time).

batch.jdbc.user=root

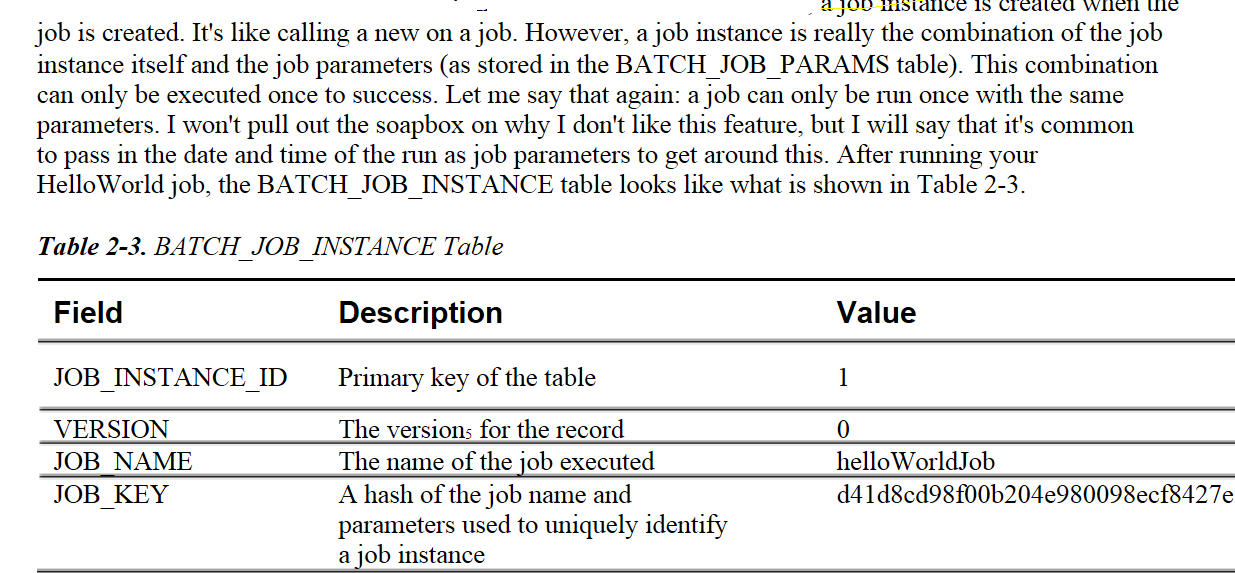
batch.jdbc.password=p@ssw0rd

batch.schema=spring\_batch\_test

#batch.schema.script=schema-mysql.sql

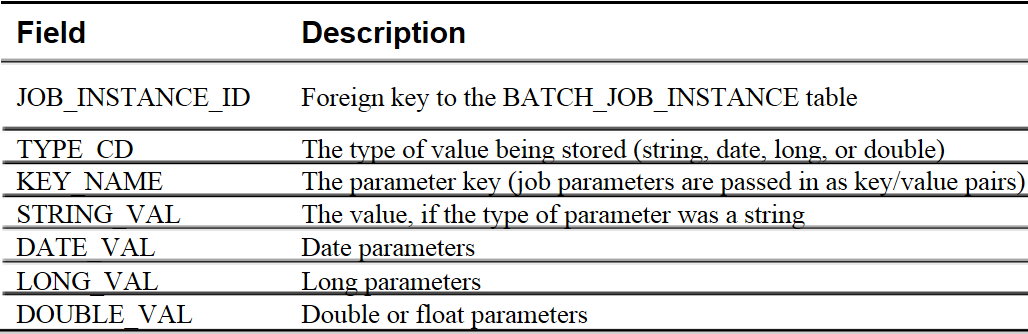
**Repository tables:**

1. **BATCH\_JOB\_INSTANCE**

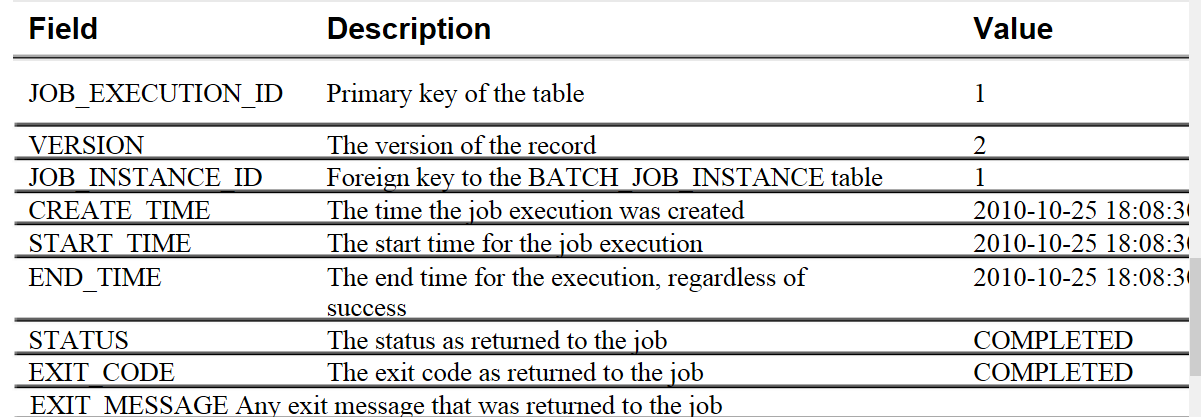
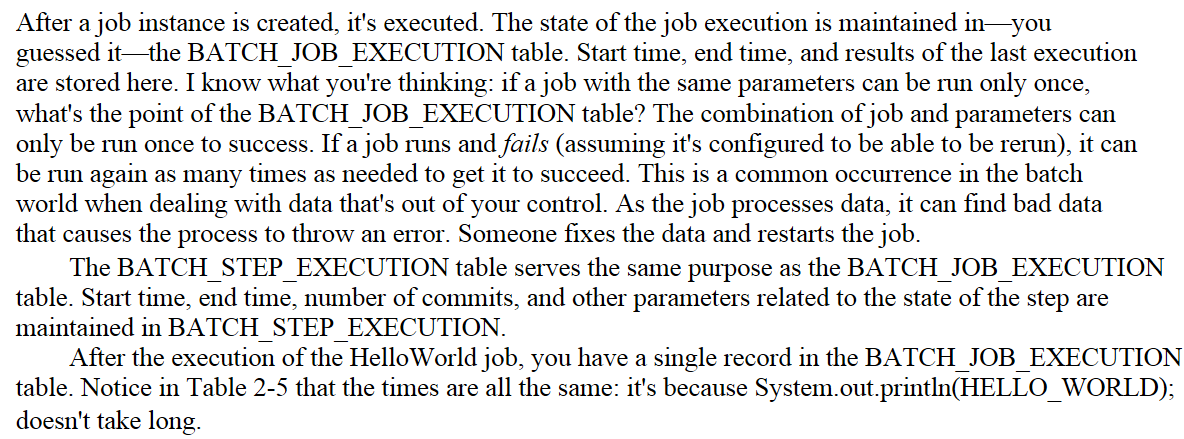


1. **BATCH\_JOB\_PARAMS**

BATCH\_JOB\_PARAMS table contains all the parameters passed to the job. As mentioned in the previous section, the parameters are part of what Spring Batch uses to identify the run of a job. In this case, the BATCH\_JOB\_PARAMS table is empty because you didn't pass any parameters to your job.

****

1. **BATCH\_JOB\_EXECUTION and BATCH\_STEP\_EXECUTION**



https://www.youtube.com/watch?v=2e4anlLoZiU&list=PLyjRfBKf4ajqMfz6yYviudRZVTSnOQSjm&index=8