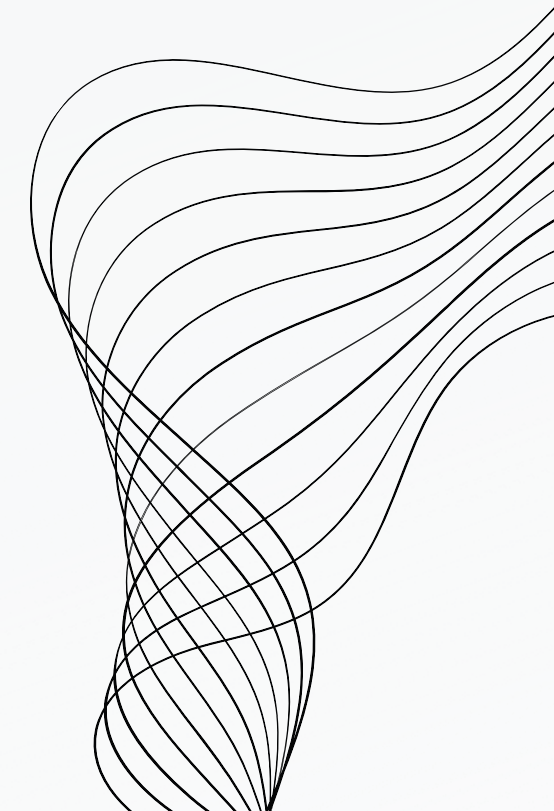


SPRING BATCH



¿QUÉ ES SPRING BATCH?



Es un framework ligero open source para procesamientos batch o procesamientos por lotes.

¿QUÉ ES UN PROCESO BATCH?



Es un proceso pensado para trabajar con volúmenes muy grandes de datos y generalmente de una forma programada. Es decir, sin intervención humana.

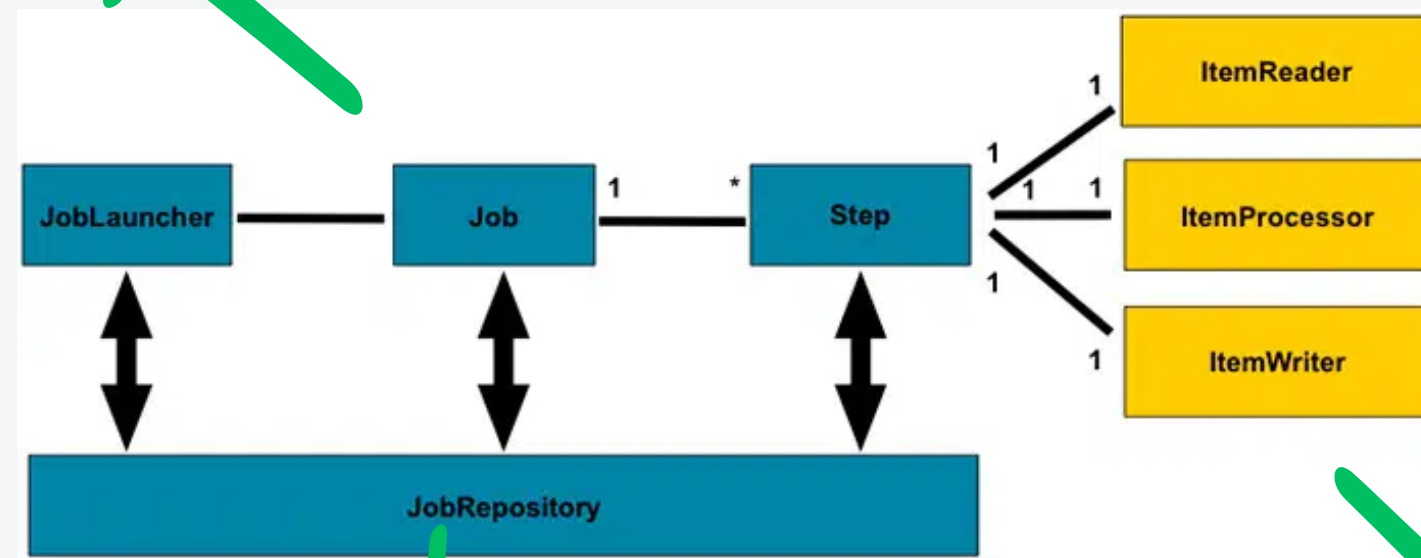
Imaginemos, por ejemplo, la carga de un fichero enorme con millones de registros; o bien un proceso nocturno que, a partir de una serie de consultas, envía una gran cantidad de e-mails, sms, etc. Esto sería un proceso batch.



COMPONENTES DE SPRING BATCH

Un job es un bloque de trabajo y está compuesto por uno o varios pasos o steps.

ItemReader: se encarga de la lectura del procesamiento por lotes. Esta lectura puede ser, por ejemplo, de una base de datos; o también podría ser de un broker de mensajes o bien un fichero csv, xml, json, etc.



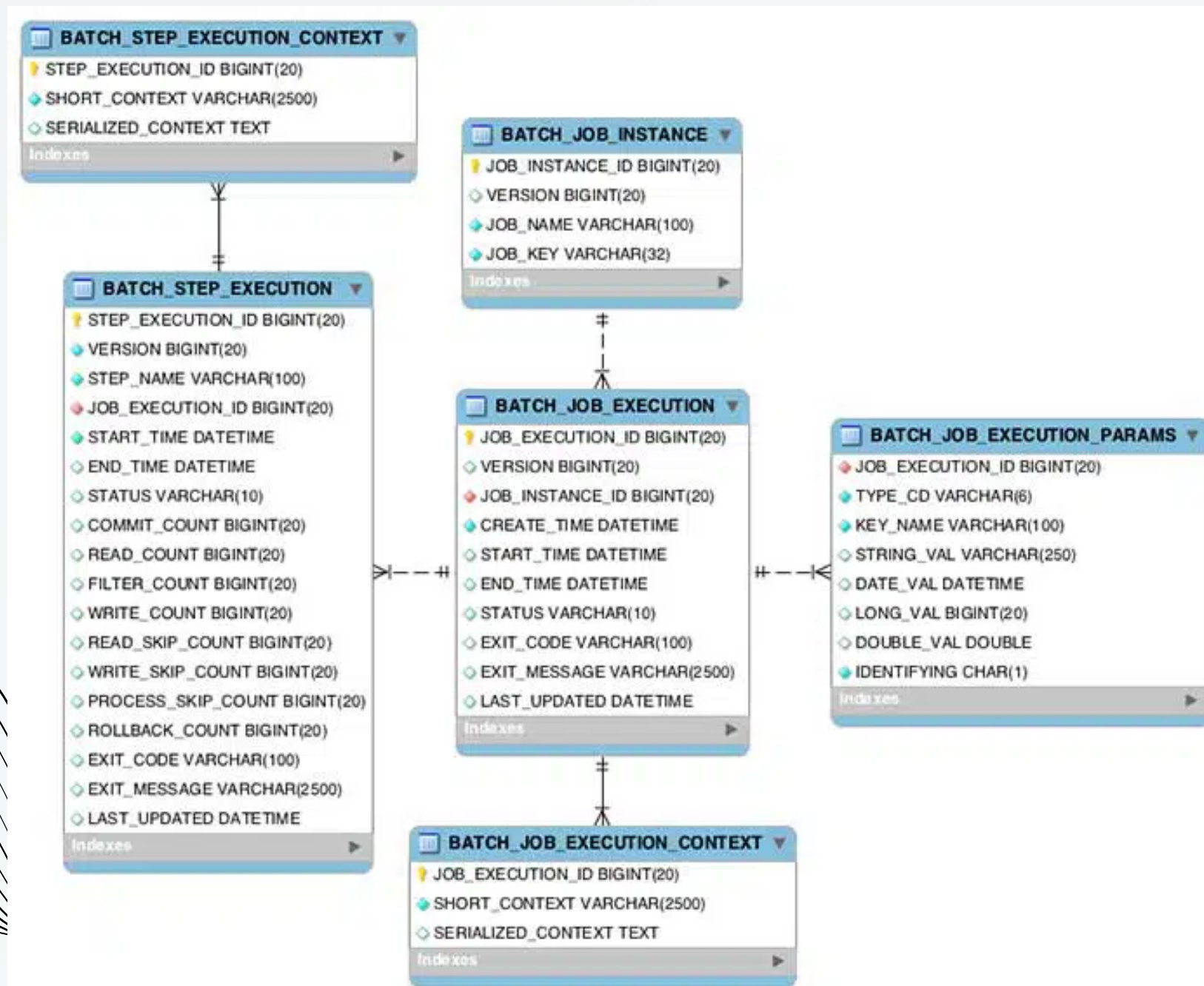
ItemProcessor: se encarga de transformar items previamente leídos. Esta transformación además de incluir cambios en el formato puede incluir filtrado de datos o lógica de negocio.

Escribe y consulta una serie de tablas existentes en base de datos. Es responsabilidad de este repositorio almacenar información sobre cada job, step que se produzca, los parámetros del Job, los errores que tengan lugar, etc.

ItemWriter: este elemento es lo opuesto al itemReader. Se encarga de la escritura de los ítems. Esta puede ser inserciones en una base de datos, en un fichero csv, en un broker de mensajes, etc.

META-DATA SCHEMA

Una vez se arranca una aplicación Spring Batch, se establece una conexión con la base de datos que contiene el esquema de tablas que utiliza el framework. Si no existe, se puede incluir por configuración que sea el propio framework el que cree el esquema de base de datos. También disponemos de la opción de cambiar el prefijo del nombre que tendrán estas tablas.



BATCH_JOB_INSTANCE

Esta tabla almacena toda la información referente a las instancias de Job.

BATCH_JOB_EXECUTION_PARAMS

En esta tabla encontraremos los parámetros que recibe cada job en formato clave/valor.

BATCH_JOB_EXECUTION

Se almacena aquí la información referente a cada ejecución del Job. Es muy útil principalmente para conocer el estado de la ejecución, si se ha completado, si ha habido errores, si aún está en ejecución. También, nos sirve para ver la duración del Job por la fecha de inicio y fecha de fin.

BATCH_STEP_EXECUTION

Esta tabla almacena información sobre cada step del job. Es muy similar a la tabla anterior, mostrándonos fecha de inicio y fin de cada step y el STATUS de cada uno de ellos. También nos indica cuántas veces ha leído, escrito y commiteado elementos cada uno de los steps de la ejecución.

EJEMPLO DE SPRING BATCH

El ejemplo consiste en una aplicación de Spring Boot con Spring Batch que realiza el siguiente proceso batch:

- Lectura de un fichero csv
- Procesamiento del fichero
- Escritura en una base de datos h2

CONFIGURACIÓN

En la clase `BatchConfiguration.java` tenemos toda la configuración del proceso batch.

```
@Configuration
@EnableBatchProcessing // Es necesario habilitar el procesamiento batch
@RequiredArgsConstructor
public class BatchConfiguration {

    private final JobBuilderFactory jobBuilderFactory;
    private final StepBuilderFactory stepBuilderFactory;

    @Bean //Mediante este bean definimos el reader de csv
    public ItemReader<CarDto> reader() {
        return new FlatFileItemReaderBuilder<CarDto>()
            .name("carItemReader")
            .resource(new ClassPathResource("sample-data.csv"))
            .linesToSkip(1)
            .delimited()
            .names(new String[]{"registration", "colour", "model", "fuelType"})
            .fieldSetMapper(new BeanWrapperFieldSetMapper<CarDto>() {{
                setTargetType(CarDto.class);
            }})
            .build();
    }
}
```

```
@Bean
public CarItemProcessor processor() {
    return new CarItemProcessor();
}

@Bean // Mediante este Bean definimos la escritura en base de datos
public ItemWriter<CarEntity> writer(DataSource dataSource) {
    return new JdbcBatchItemWriterBuilder<CarEntity>()
        .itemSqlParameterSourceProvider(new BeanPropertyItemSqlParameterSourceProvider<>())
        .sql("INSERT INTO cars (id, registration, colour, model, fuelType ) VALUES (:id, :registration, " +
            ":colour, :model, :fuelType)")
        .dataSource(dataSource)
        .build();
}
```

```
@Bean // Aquí se define el Job
public Job createEmployeeJob(CarJobExecutionListener listener, Step step1) {
    return jobBuilderFactory
        .get("createEmployeeJob")
        .incrementer(new RunIdIncrementer())
        .listener(listener)
        .flow(step1) // Se podrían incluir múltiples steps
        .end()
        .build();
}

@Bean // Aquí se define el step, observamos que se le pasan como parametros reader, writer y processor
public Step step1(ItemReader<CarDto> reader, ItemWriter<CarEntity> writer,
    ItemProcessor<CarDto, CarEntity> processor) {
    return stepBuilderFactory
        .get("step1")
        .<CarDto, CarEntity>chunk(2) // Es posible parameterizar el tamaño de chunk
        .reader(reader)
        .processor(processor)
        .writer(writer)
        .build();
}
```

EJEMPLO DE SPRING BATCH

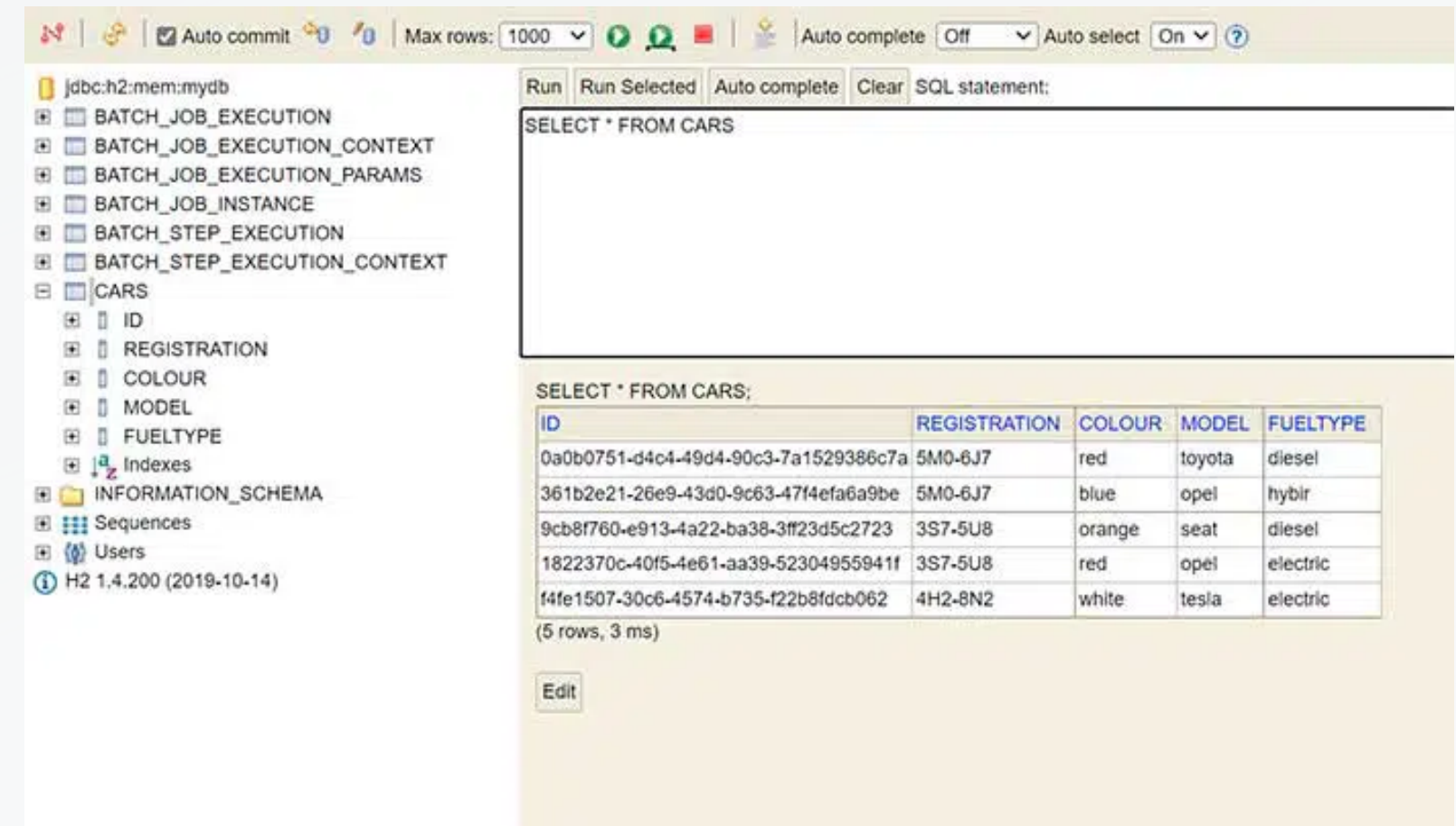
- Observamos que hemos incluido en el job un listener. Esto nos permitiría estar a la escucha del comienzo o el fin del job, como vemos a continuación:

```
@Component
@RequiredArgsConstructor
@Slf4j
public class CarJobExecutionListener implements JobExecutionListener {

    @Override
    public void beforeJob(JobExecution jobExecution) {
        log.info("Executing cars job with id {}", jobExecution.getId());
    }

    @Override
    public void afterJob(JobExecution jobExecution) {
        if(jobExecution.getStatus() == BatchStatus.COMPLETED) {
            log.info("Cars job with id {} execution completed", jobExecution.getId());
        }
    }
}
```

Finalmente si deseamos comprobar si la carga se ha efectuado correctamente, si levantamos la aplicación en nuestro local podemos acceder al h2 y observar la carga que se ha realizado, así como las tablas de configuración del proceso batch.



The screenshot shows the H2 database console interface. On the left, a tree view displays the database schema, including tables like BATCH_JOB_EXECUTION, BATCH_JOB_EXECUTION_CONTEXT, BATCH_JOB_EXECUTION_PARAMS, BATCH_JOB_INSTANCE, BATCH_STEP_EXECUTION, BATCH_STEP_EXECUTION_CONTEXT, CARS, INFORMATION_SCHEMA, Sequences, and Users. The 'CARS' table is selected. On the right, the SQL statement 'SELECT * FROM CARS;' is entered in the 'SQL statement:' field. Below the field, the results of the query are displayed in a table with 5 rows and 5 columns: ID, REGISTRATION, COLOUR, MODEL, and FUELTYPE. The data shows five cars: a red Toyota diesel, a blue Opel hybrid, an orange Seat diesel, a red Opel electric, and a white Tesla electric. The interface also includes buttons for 'Run', 'Run Selected', 'Auto complete', 'Clear', and 'Edit', as well as a status bar indicating '(5 rows, 3 ms)'.

ID	REGISTRATION	COLOUR	MODEL	FUELTYPE
0a0b0751-d4c4-49d4-90c3-7a1529386c7a	5M0-6J7	red	toyota	diesel
361b2e21-26e9-43d0-9c63-47f4efa6a9be	5M0-6J7	blue	opel	hybir
9cb8f760-e913-4a22-ba38-3ff23d5c2723	3S7-5U8	orange	seat	diesel
1822370c-40f5-4e61-aa39-52304955941f	3S7-5U8	red	opel	electric
f4fe1507-30c6-4574-b735-f22b8fdbc062	4H2-8N2	white	tesla	electric

PRINCIPIOS A LA HORA DE DEFINIR UN PROCESO BATCH

- Simplificar todo lo posible la lógica: de forma que quede fragmentada en procesos muy pequeños de lectura, procesamiento y escritura.
- Utilizar los mínimos recursos posibles: ya que se va a procesar un enorme volumen de datos.
- Revisar y optimizar sentencias sql: es esencial que las consultas estén optimizadas ya que se verá tanto en el redimiendo de la base de datos como en los tiempos de ejecución del job.
- Utilizar comprobaciones checksum: esto es muy útil a la hora de generar ficheros de gran tamaño, incluir en el pie del fichero un contador con el numero de registros e información del procesamiento realizado que puede ayudar a comprobar la integridad del fichero.
- Utilizar pruebas de stress con datos lo más realistas posibles.

GRACIAS!!



Spring Batch

