Xideral

Back End Academy

Spring Batch

Project By:

Kenneth Rodríguez García

Spring Batch

Summary

This project is a demonstration of the Spring Batch framework and how it handles large volumes of data with batch processing. Our application will use a movie database and migrate selected data from a csv file to a MySQL database.

Research

Batch Processing

Batch processing is a technique used to process large volumes of data in a single run, which is usually scheduled or automated.

If we compare batch processing to a traditional REST API, in a REST API data is handled as soon as it is received; on the other hand, batch processing might temporarily store the data to be processed later.

A single batch usually consists of a large dataset, and the processing is often done without any user interaction.

The most common use cases for batch processing are:

- Data Migration
- ETL (Extract, Transform, Load)
- Report Generation

Batch processing provides several advantages, such as great cost-effectiveness when handling certain types of workloads, as well as processing large amounts of data quickly and efficiently.

However, its biggest disadvantage is the delay between the time data is received and the time it is processed. Depending on the application setup, the delay might be hours, or even days long.

Spring Batch

Spring Batch is a framework that handles batch processing applications, which helps developers handle large volumes of data.

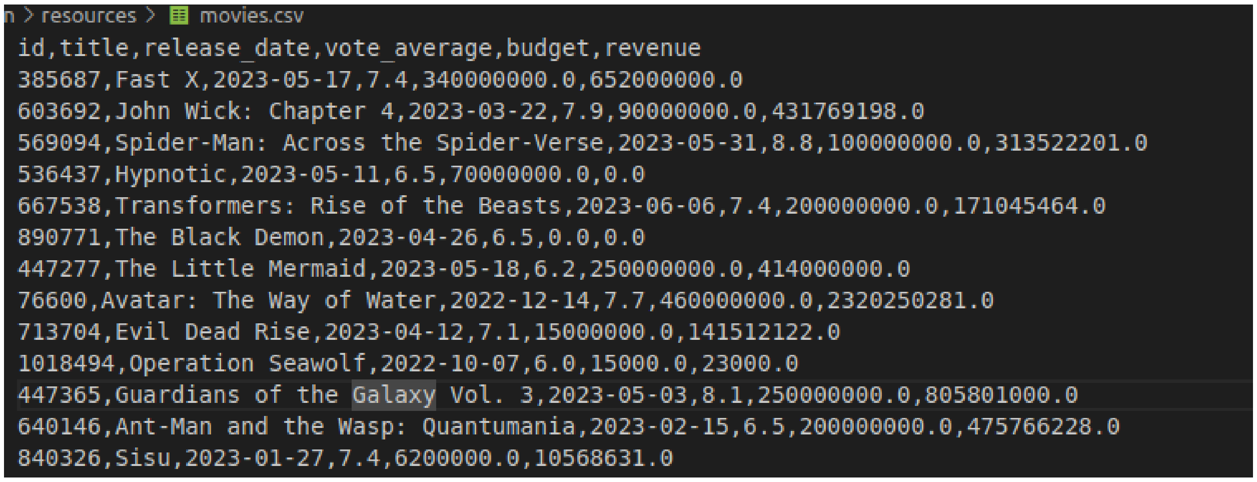
The framework allows developers to define “jobs” which are themselves composed of smaller “steps”. Each step represents a specific transformation or process to the batch. This allows developers to define multiple independent steps and build jobs by selecting and ordering these steps.

When the workload is too big to process at once, Spring Batch allows for processing the data in “chunks”, which are smaller subsets of data. While this might slow down the processing time, it can improve the performance and memory usage of an application.

The framework also provides tools to restart and recover failed jobs, which ensures that the process is completed successfully and no data is lost.

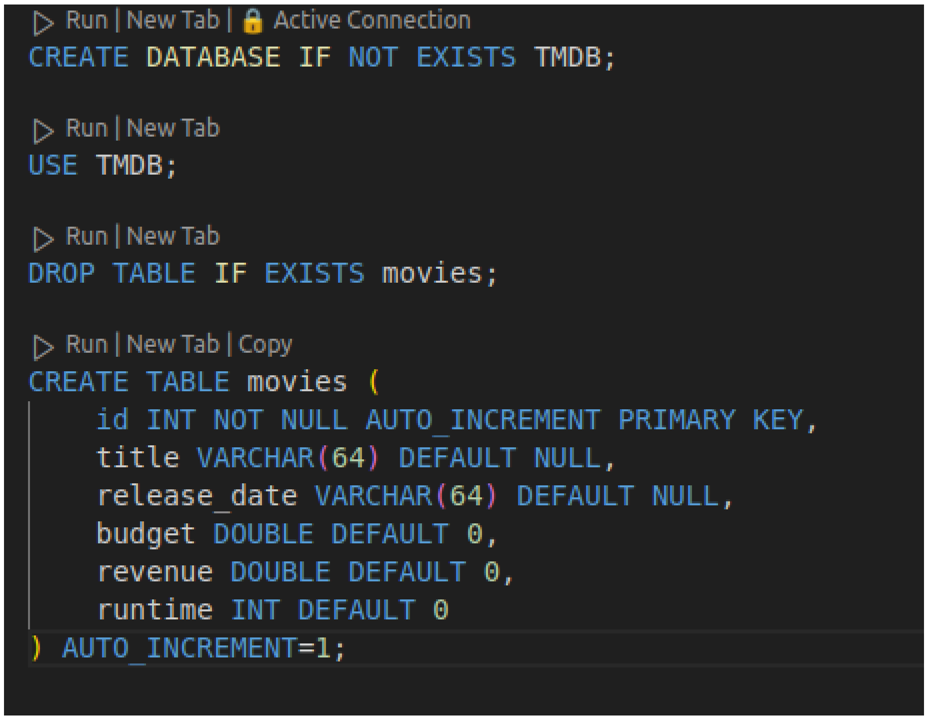
Design

Our .csv file will contain data from over 7000 different movies, allowing us to properly test Spring Batch's functionalities. The .csv file, aptly named **movies.csv**, contains the title, release date, vote average (score), budget, and revenue information for each of the different entries.



```
id,title,release_date,vote_average,budget,revenue
385687,Fast X,2023-05-17,7.4,340000000.0,652000000.0
603692,John Wick: Chapter 4,2023-03-22,7.9,90000000.0,431769198.0
569094,Spider-Man: Across the Spider-Verse,2023-05-31,8.8,100000000.0,313522201.0
536437,Hypnotic,2023-05-11,6.5,70000000.0,0.0
667538,Transformers: Rise of the Beasts,2023-06-06,7.4,200000000.0,171045464.0
890771,The Black Demon,2023-04-26,6.5,0.0,0.0
447277,The Little Mermaid,2023-05-18,6.2,250000000.0,414000000.0
76600,Avatar: The Way of Water,2022-12-14,7.7,460000000.0,2320250281.0
713704,Evil Dead Rise,2023-04-12,7.1,15000000.0,141512122.0
1018494,Operation Seawolf,2022-10-07,6.0,15000.0,23000.0
447365,Guardians of the Galaxy Vol. 3,2023-05-03,8.1,250000000.0,805801000.0
640146,Ant-Man and the Wasp: Quantumania,2023-02-15,6.5,200000000.0,475766228.0
840326,Sisu,2023-01-27,7.4,6200000.0,10568631.0
```

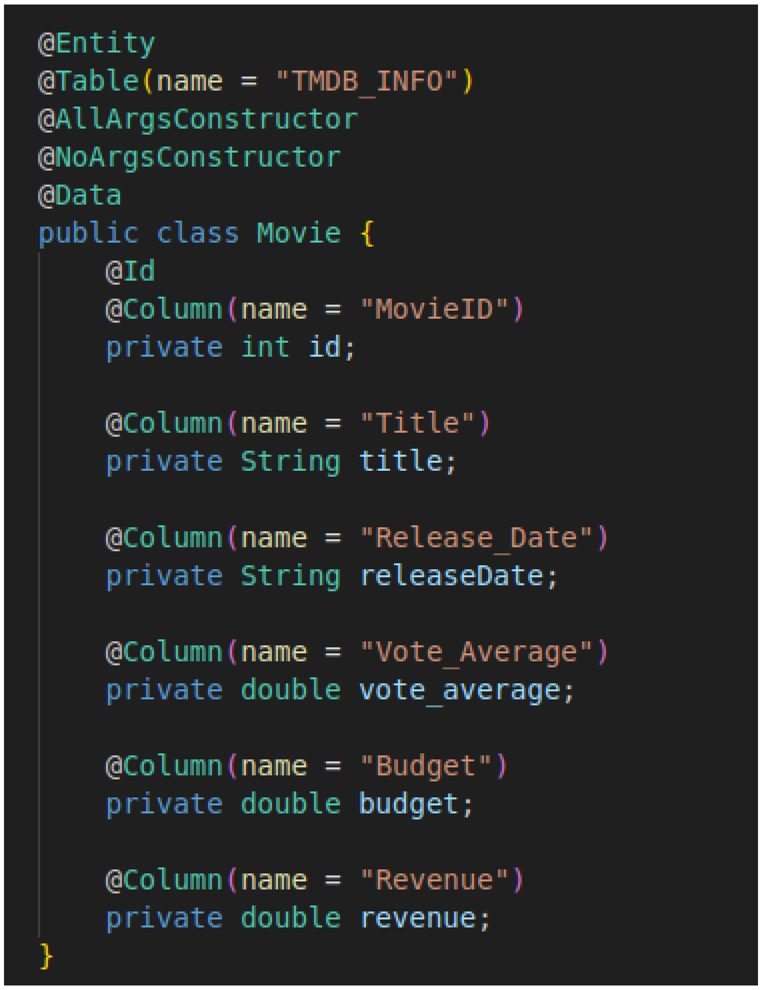
The application will migrate all data from this .csv file to a MySQL database, so we need to prepare the corresponding table to properly store the data.



```sql
CREATE DATABASE IF NOT EXISTS TMDB;

USE TMDB;

DROP TABLE IF EXISTS movies;

CREATE TABLE movies (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    title VARCHAR(64) DEFAULT NULL,
    release_date VARCHAR(64) DEFAULT NULL,
    budget DOUBLE DEFAULT 0,
    revenue DOUBLE DEFAULT 0,
    runtime INT DEFAULT 0
) AUTO_INCREMENT=1;
```

Entity - Movie

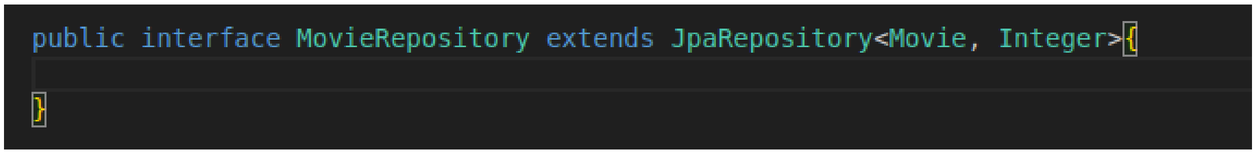
The Movie class is the basic entity of the program. Each Movie object represents an entry or row in the database.



```java
@Entity
@Table(name = "TMDB_INFO")
@AllArgsConstructor
@NoArgsConstructor
@Data
public class Movie {
    @Id
    @Column(name = "MovieID")
    private int id;

    @Column(name = "Title")
    private String title;

    @Column(name = "Release_Date")
    private String releaseDate;

    @Column(name = "Vote_Average")
    private double vote_average;

    @Column(name = "Budget")
    private double budget;

    @Column(name = "Revenue")
    private double revenue;
}
```

Repository - MovieRepository

MovieRepository uses the Spring Data JPA framework to provide persistence to the application. This will allow data to be properly uploaded to the database.



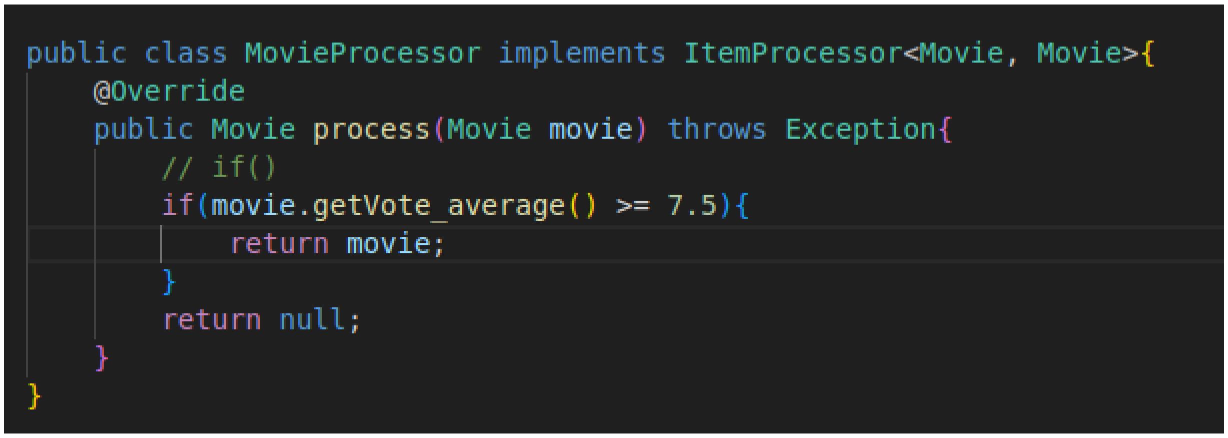
```java
public interface MovieRepository extends JpaRepository<Movie, Integer>{

}
```

By inheriting from JpaRespository, the framework provides the MovieRepository class with lots of functionality.

ItemProcessor - MovieProcessor

This class is used to process the data entities, and essentially works as a filter or validator for our data. This can be used to filter entities with missing data or that do not fit a specific requirement.

For this project, we will only use movies that have a score equal to or higher than 7.5, so we process accordingly.



```
public class MovieProcessor implements ItemProcessor<Movie, Movie>{
    @Override
    public Movie process(Movie movie) throws Exception{
        // if()
        if(movie.getVote_average() >= 7.5){
            return movie;
        }
        return null;
    }
}
```

Spring Batch Configuration - SpringBatchConfig

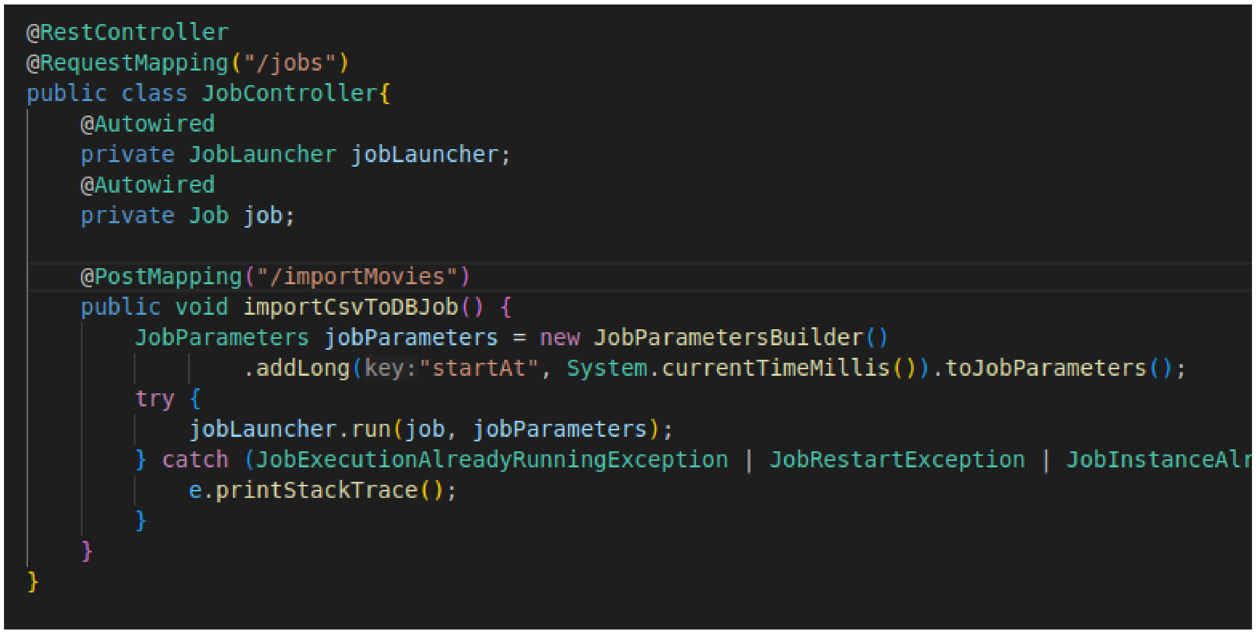
The SpringBatchConfig class provides the definition and implementation for our batch application. It is in this file where we can configure the different steps and jobs we want our application to execute.

Focusing on these methods, we have the ***step1()*** method, which reads the .csv file, processes the data, and writes it to the database. On the other hand we have the ***runJob()*** method, which will actually execute the *step1()* method, as it should not be directly executed.

Note that our *Job* only executes a single *Step*. In practice, a *Job* can perform any number of *Steps* in sequence, but since our application only defines a single *Step* there are no more to execute.

Controller - JobController

The JobController class will handle the execution of our *Job*. It is a RestController, which means that a REST API will be how the job execution is triggered.



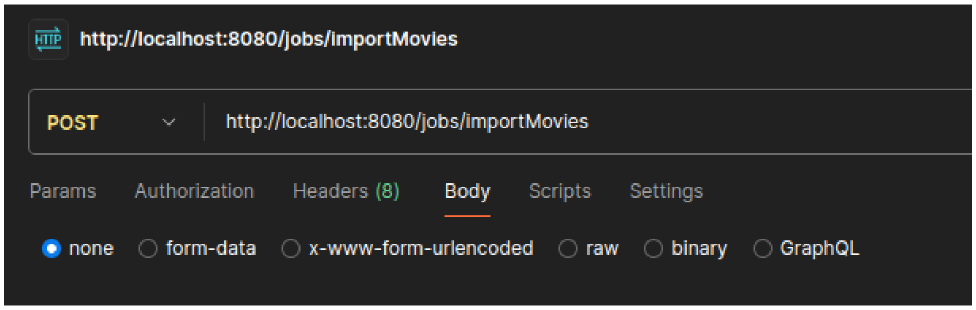
```
@RestController
@RequestMapping("/jobs")
public class JobController{
    @Autowired
    private JobLauncher jobLauncher;
    @Autowired
    private Job job;

    @PostMapping("/importMovies")
    public void importCsvToDBJob() {
        JobParameters jobParameters = new JobParametersBuilder()
                .addLong(key:"startAt", System.currentTimeMillis()).toJobParameters();
        try {
            jobLauncher.run(job, jobParameters);
        } catch (JobExecutionAlreadyRunningException | JobRestartException | JobInstanceAlr
            e.printStackTrace();
        }
    }
}
```

Because our task is data migration and we will be uploading data to a database, our only API endpoint will be a POST method.

Because the process might take a while to finish execution, we must handle relevant exceptions, like checking if a Job isn't already running, or a Job's parameters are invalid.

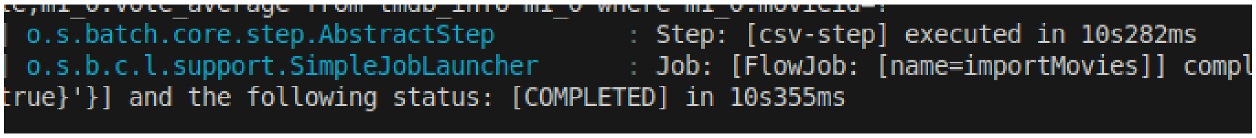
Execution

After running the Spring application, we can use **Postman** to execute the data migration.



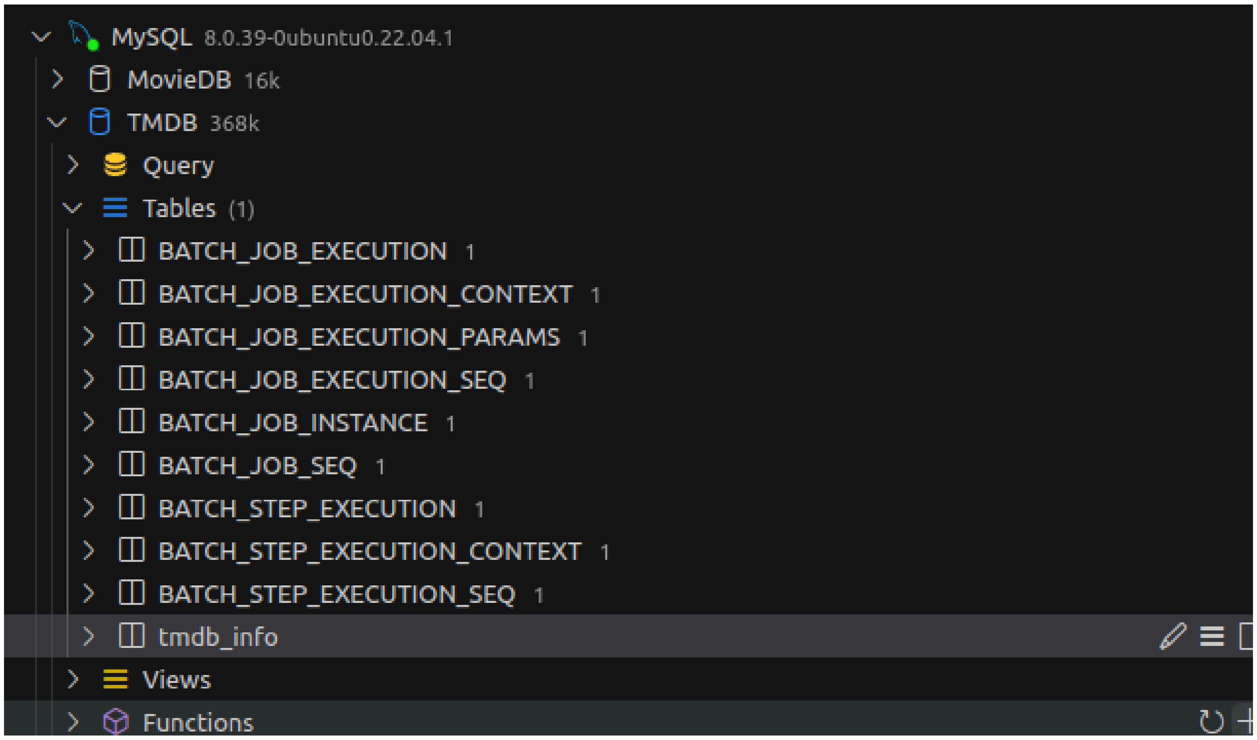
```
http://localhost:8080/jobs/importMovies

POST  http://localhost:8080/jobs/importMovies

Params  Authorization  Headers (8)  Body  Scripts  Settings

● none  ○ form-data  ○ x-www-form-urlencoded  ○ raw  ○ binary  ○ GraphQL
```

The execution returned a successful HTTP Status Code, which means the execution was successful. We can also verify by reading the console output of the Spring application, which displays a "COMPLETED" message, and the total execution time.



```
e,mi_o.vote_average from tmdb_info mi_o where mi_o.movieid=?
| o.s.batch.core.step.AbstractStep         : Step: [csv-step] executed in 10s282ms
| o.s.b.c.l.support.SimpleJobLauncher      : Job: [FlowJob: [name=importMovies]] compl
true}'}] and the following status: [COMPLETED] in 10s355ms
```

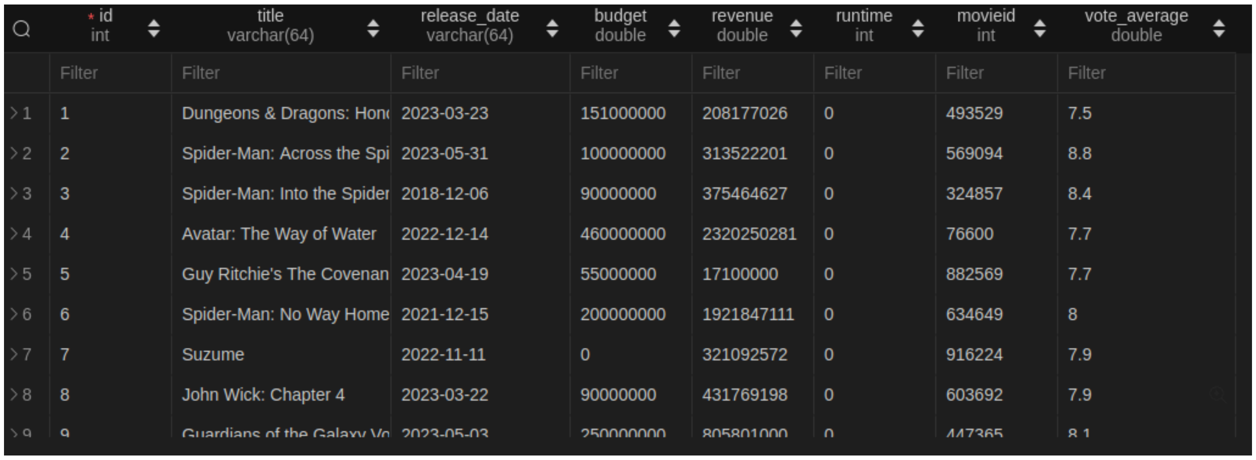
If we use a Database Management System, we can observe how the database has been updated.



```
MySQL 8.0.39-0ubuntu0.22.04.1
  MovieDB 16k
  TMDB 368k
    Query
    Tables (1)
      BATCH_JOB_EXECUTION 1
      BATCH_JOB_EXECUTION_CONTEXT 1
      BATCH_JOB_EXECUTION_PARAMS 1
      BATCH_JOB_EXECUTION_SEQ 1
      BATCH_JOB_INSTANCE 1
      BATCH_JOB_SEQ 1
      BATCH_STEP_EXECUTION 1
      BATCH_STEP_EXECUTION_CONTEXT 1
      BATCH_STEP_EXECUTION_SEQ 1
      tmdb_info
    Views
    Functions
```

Not only has the “TMDB_INFO” table been updated, but we now have a full history of the batch process execution.

A simple query can now display the entries in our table:



	* id int	title varchar(64)	release_date varchar(64)	budget double	revenue double	runtime int	movieid int	vote_average double
	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter
> 1	1	Dungeons & Dragons: Hon	2023-03-23	151000000	208177026	0	493529	7.5
> 2	2	Spider-Man: Across the Spi	2023-05-31	100000000	313522201	0	569094	8.8
> 3	3	Spider-Man: Into the Spider	2018-12-06	90000000	375464627	0	324857	8.4
> 4	4	Avatar: The Way of Water	2022-12-14	460000000	2320250281	0	76600	7.7
> 5	5	Guy Ritchie's The Covenan	2023-04-19	55000000	17100000	0	882569	7.7
> 6	6	Spider-Man: No Way Home	2021-12-15	200000000	1921847111	0	634649	8
> 7	7	Suzume	2022-11-11	0	321092572	0	916224	7.9
> 8	8	John Wick: Chapter 4	2023-03-22	90000000	431769198	0	603692	7.9
> 9	9	Guardians of the Galaxy Vo	2023-05-03	250000000	805801000	0	447365	8.1

Conclusion

Batch processing is one of those topics I already had an intuitive idea about, but now I am lucky to say I’ve finally formally learned about it.

Although I still have some trouble fully accepting the Step and Job architecture of batch processing, I can also appreciate their usefulness when working with large amounts of data.

Despite already knowing about Spring Web, and recently learning about Spring Data JPA, I still feel amazed (almost overwhelmed) by the entire Spring ecosystem. As a developer I can appreciate how much these tools and frameworks really improve the development process.