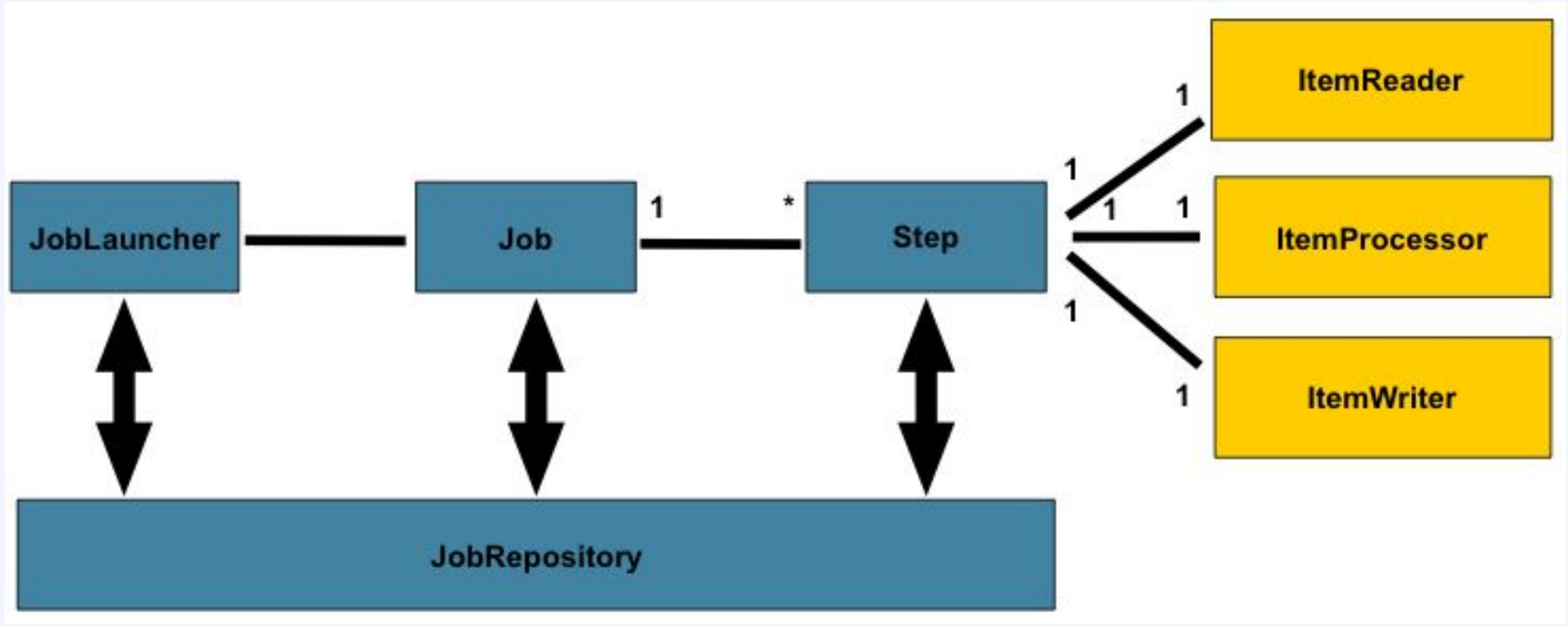
Spring Batch 아키텍처

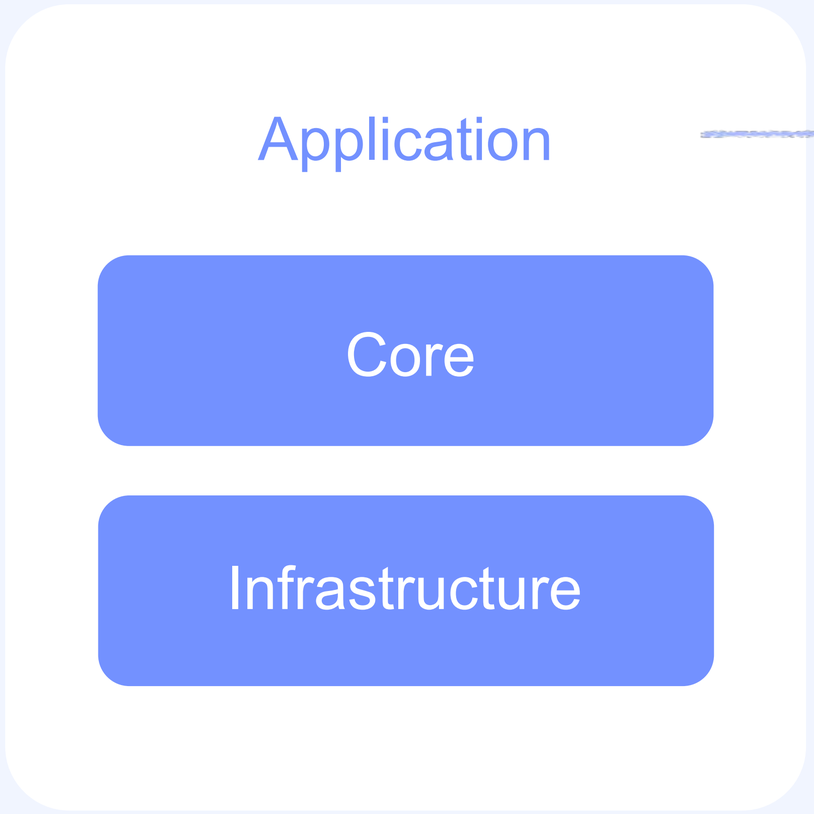
1 Job

Spring Batch 아키텍처

Spring Batch 도메인 언어

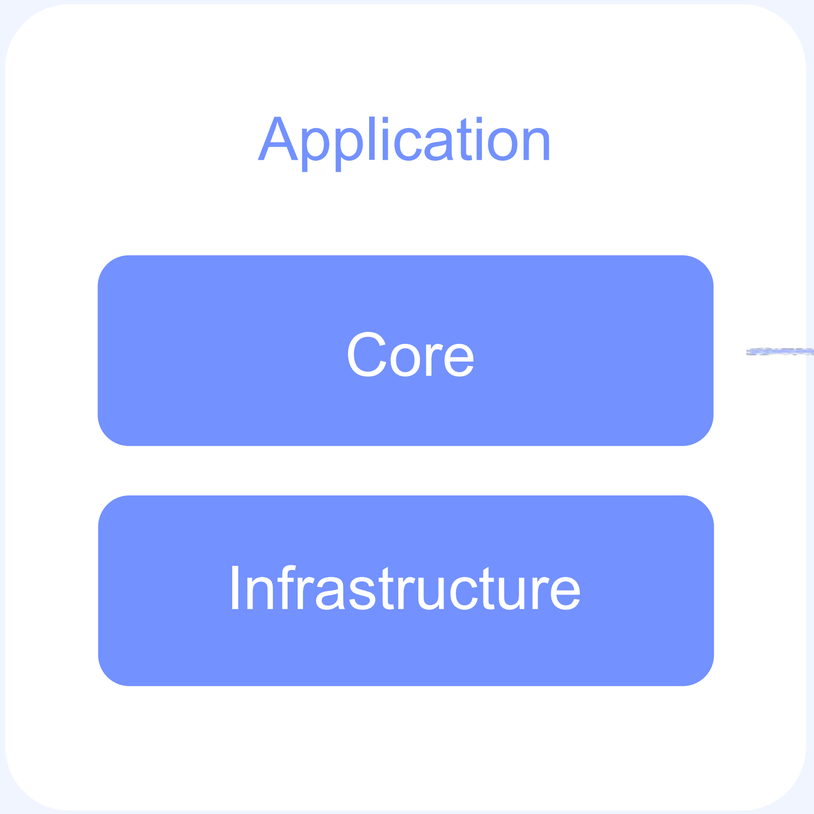


- JobLauncher는 Job을 실행시키는 컴포넌트
- Job은 배치 작업. JobRepository는 Job 실행과 Job, Step을 저장
- Step은 배치 작업의 단계. ItemReader, ItemProcessor, ItemWriter는 데이터를 읽고 처리하고 쓰는 구성



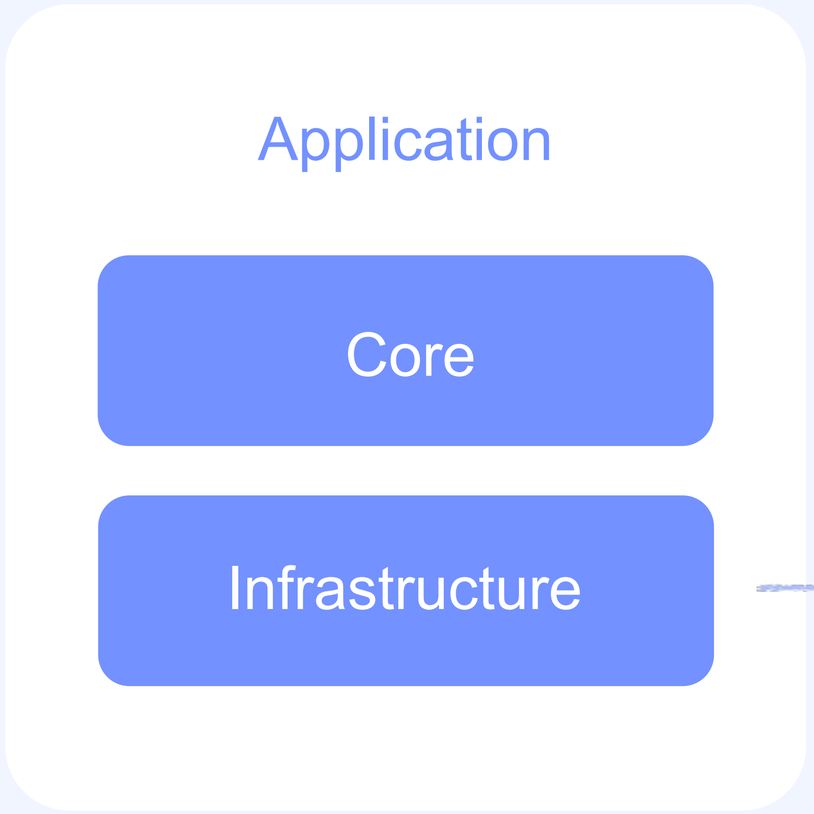
Application Layer

- 사용자(=우리) 코드와 구성
- 비즈니스, 서비스 로직
- Core, Infrastructure를 이용해 배치의 기능을 만든다



Core Layer

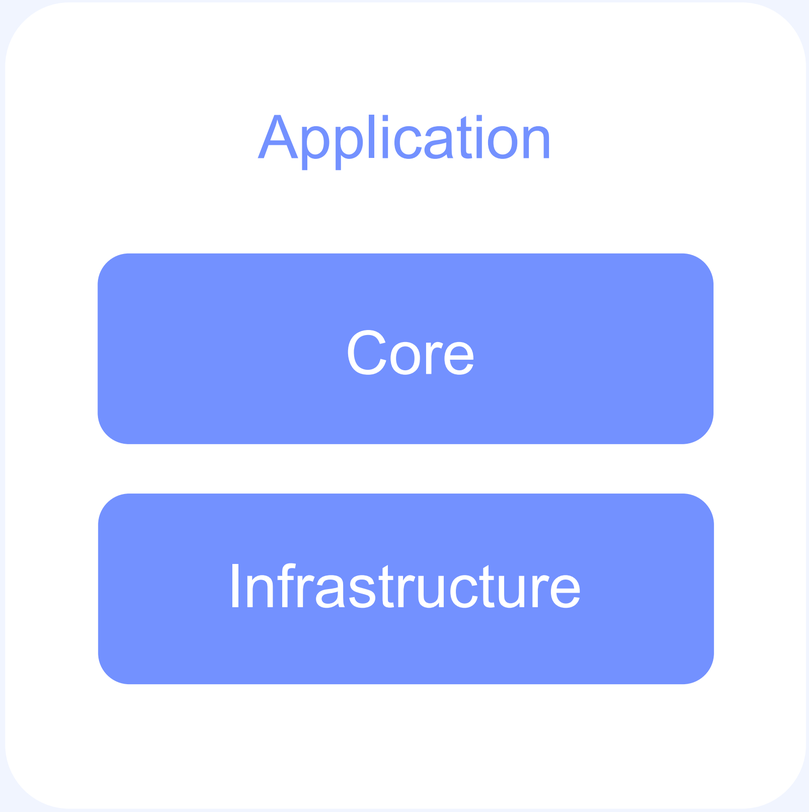
- 배치 작업을 시작하고 제어하는데 필수적인 클래스
- Job, Step, JobLaucher



Infrastructure Layer

- 외부와 상호작용
- ItemReader, ItemWriter, RetryTemplate

Spring Batch 아키텍처

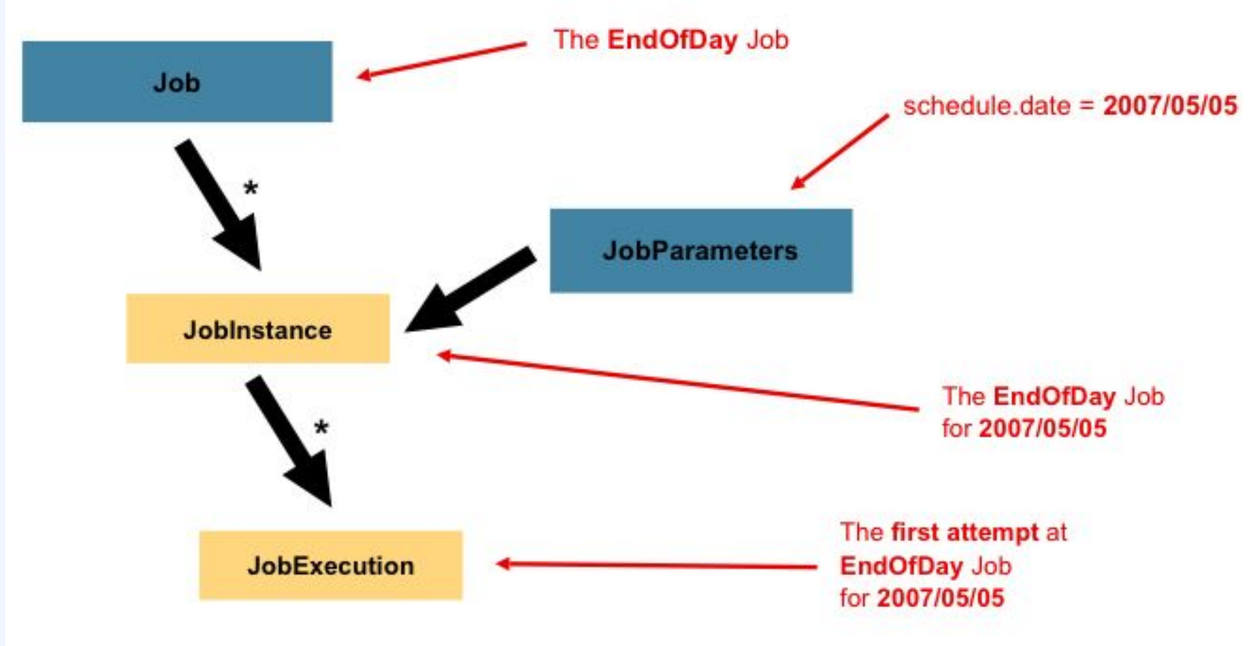


Spring Batch가 제공하는 Core와 Infrastructure를 활용해 Application을 구현한다

Job

Job (일)

- 전체 배치 프로세스를 캡슐화한 도메인
- Step의 순서를 정의한다.
- JobParameters 받는다.



Job (일)

- 전체 배치 프로세스를 캡슐화한 도메인
- Step의 순서를 정의한다.
- JobParameters 받는다.

Code

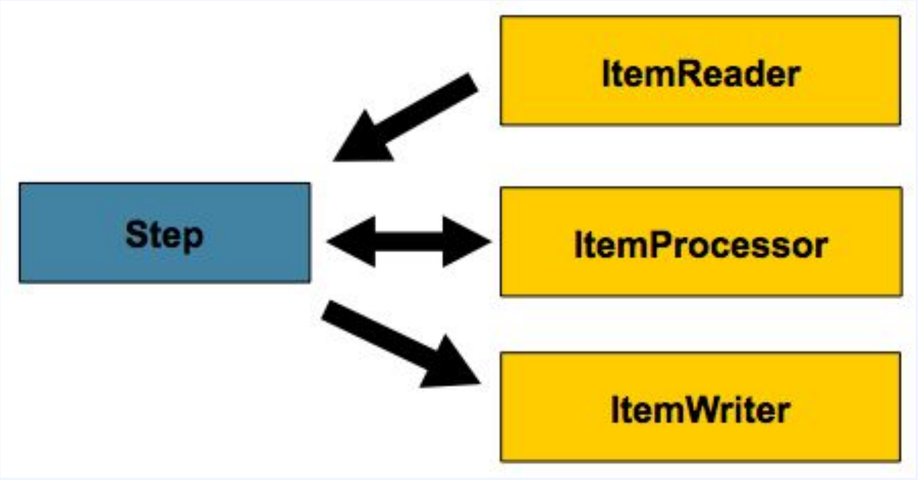
```
@Bean
public Job footballJob() {
    return this.jobBuilderFactory.get("footballJob")
                    .start(playerLoad())
                    .next(gameLoad())
                    .next(playerSummarization())
                    .build();
}
```

Spring Batch 아키텍처

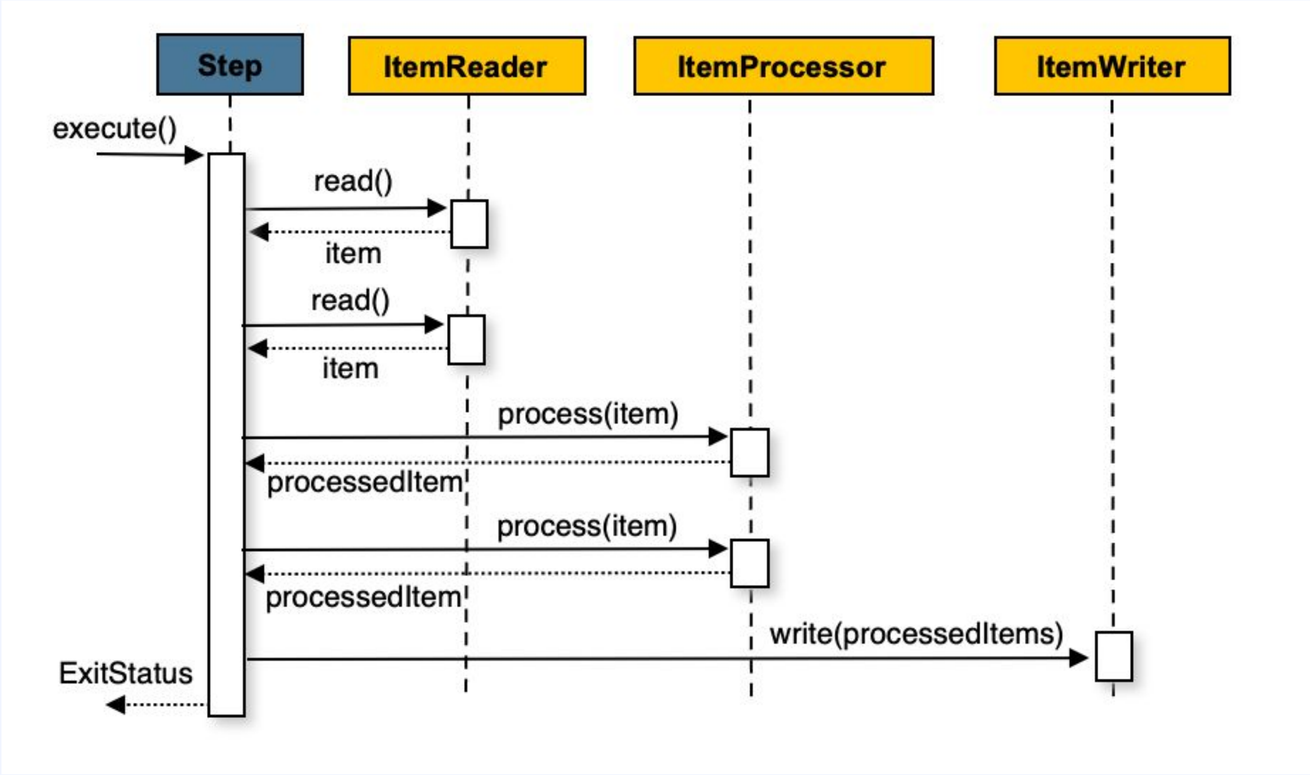
2 Step

Step

- 작업 처리의 단위
- Chunk 기반 스텝, Tasklet 스텝 2가지로 나뉜다.



Chunk-oriented Processing Step



chunk기반으로 하나의 트랜잭션에서 데이터를 처리한다.

commitInterval만큼 데이터를 읽고 트랜잭션 경계 내에서 chunkSize만큼 write를 한다.

Chunk-oriented Processing Step

```
List items = new Arraylist();
for(int i = 0; i < commitInterval; i++){
    Object item = itemReader.read();
    if (item != null) {
        items.add(item);
    }
}

List processedItems = new Arraylist();
for(Object item: items){
    Object processedItem = itemProcessor.process(item);
    if (processedItem != null) {
        processedItems.add(processedItem);
    }
}

itemWriter.write(processedItems);
```

- chunkSize: 한 트랜잭션에서 쓸 아이템의 갯수
- commitInterval: reader가 한번에 읽을 아이템의 갯수
- chunkSize >= commitInterval 하지만 보통 같게 맞춰서 사용하는 것이 좋다.

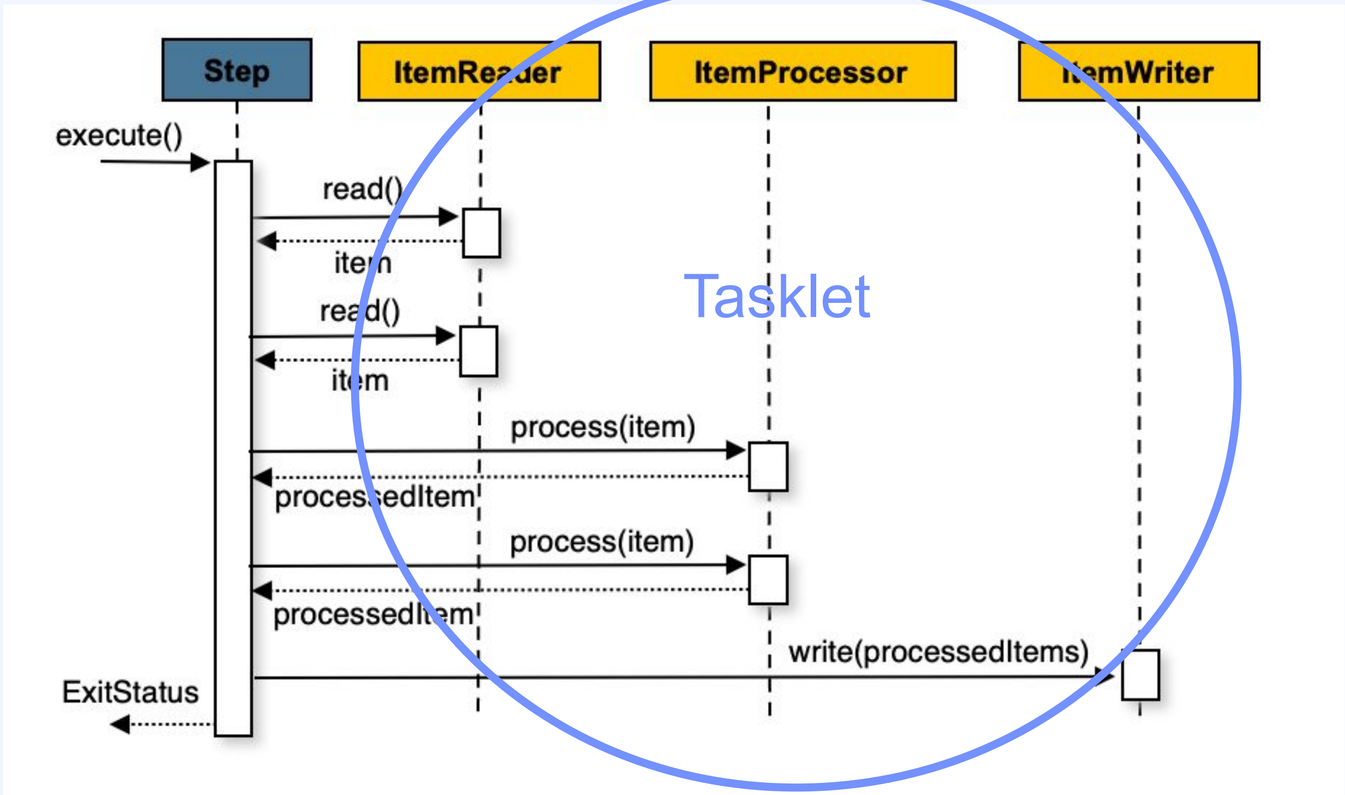
Chunk-oriented Processing Step

```
@Bean
public Job sampleJob(JobRepository jobRepository, Step sampleStep) {
    return this.jobBuilderFactory.get("sampleJob")
          .repository(jobRepository)
                .start(sampleStep)
                .build();
}

@Bean
public Step sampleStep(PlatformTransactionManager transactionManager) {
  return this.stepBuilderFactory.get("sampleStep")
        .transactionManager(transactionManager)
        .<String, String>chunk(10)
        .reader(itemReader())
        .writer(itemWriter())
        .build();
}
```

- ItemReader, ItemProcessor, ItemWriter 구현체를 설정한다.
- ItemProcessor는 생략할 수 있다.

TaskletStep



하나의 트랜잭션에서 데이터를 처리한다.

단순한 처리를 할 때 사용한다.

TaskletStep


```
@Bean
public Step step1() {
    return this.stepBuilderFactory.get("step1")
            .tasklet(myTasklet())
            .build();
}
```

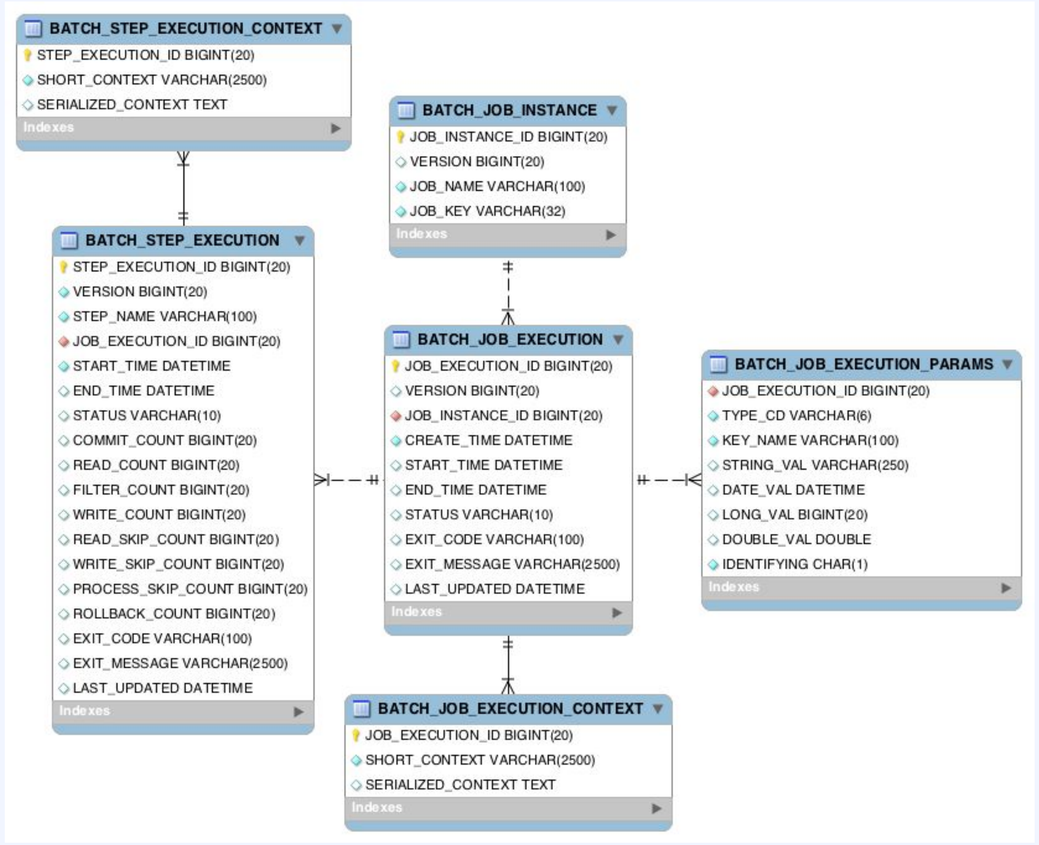
- Tasklet 구현체를 설정한다. 내부에 단순한 읽기, 쓰기, 처리 로직을 모두 넣는다.
- RepeatStatus (반복 상태)를 설정한다. RepeatStatus.FINISHED

Spring Batch 아키텍처

3 Spring Batch 스키마 구조

Spring Batch 스키마 구조

Meta Data Schema

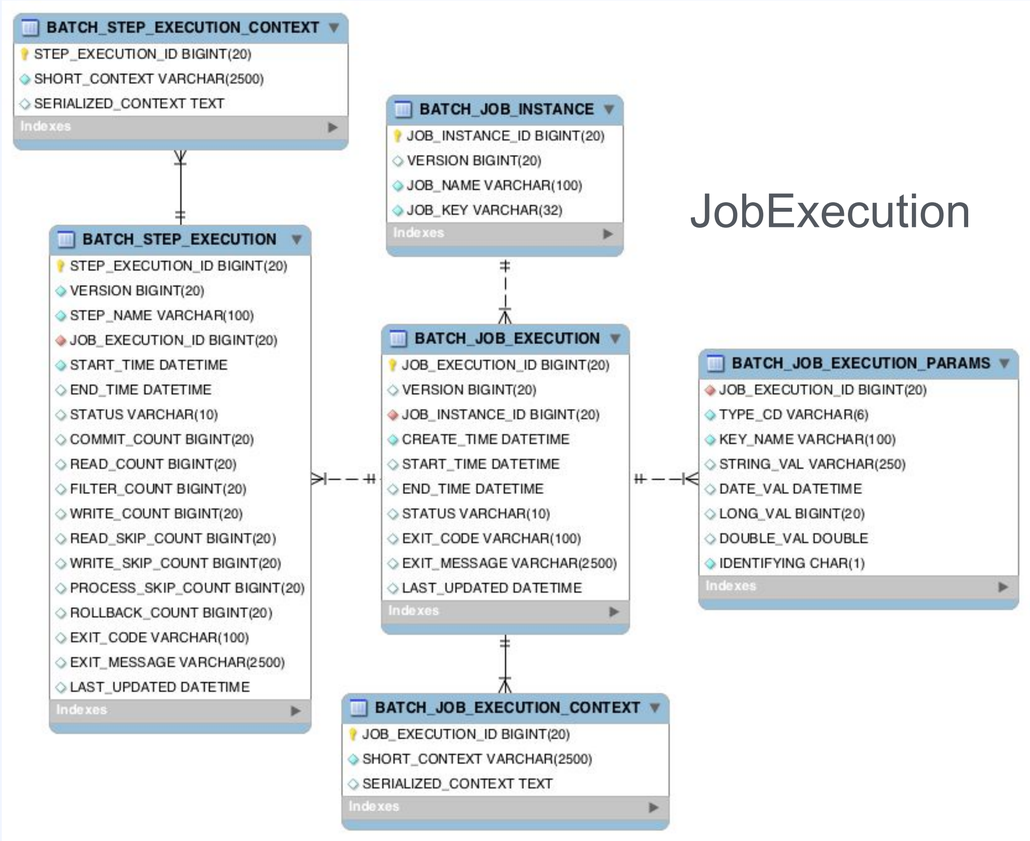


배치 실행하고 관리하기 위한 메타 데이터가 저장된다.

Spring Batch 스키마 구조

JobInstance

StepExecution



JobExecution

JobParameters

ExectionContext.Step

ExectionContext.Job

Meta Data 스키마 활용하기

- Spring Batch Framework가 실행 시 meta data 테이블들을 사용하므로 초기 설정이 필요하다.
- Spring Batch Framework에 속하는 부분이므로 수정하지 않고 조회만 한다.
- Job의 이력, 파라미터 등 실행 결과를 조회할 수 있다.
- 배치 결과에 대해서 로그, 별도의 실행 이력을 남기는 경우가 대부분이므로 조회할 일이 많지 않다.